विशाल जाधव सरांचे

VJTech Academy

Inspiring Your Success...

“Java Programming Language”

****
* Class Name : VJTech Academy , Maharashtra   *
* Author Name: Vishal Jadhav Sir   *
* Mobile No : 9730087674   *
****

==========
UNIT-I :Basic Syntactical constructs in Java
==========

Features of OOP:
==========
1) Emphasis is on data rather than procedure.
2) Programs are divided into Objects.
3) Data is hidden and can not accessed from the external functions.
4) Objects may communicate with each other through functions.
5) New data and functions can be added easily whenever required.
6) OOP follow bottom-up approach in program design.
7) Object is a collection of data & functions, function operates on data.
8) More security provided for the data compare to POP.
9) Data cannot move openely from one function to another function.

==========
Basic concepts of OOP:
==========
1) Objects:
- Basic runtime entities known as object.
- They may represent person, table, bank account or any item that program may handle.
- Object is a collection of data and functions.
- Objects are created from class.

2) Classes:
- It is a collection of similiar types of objects.
- Class is a collection of data and functions, functions operate on data.
- You may creates objects from the class.
- When you define the class then memory will not be allocated for the members.
- Class shows data abstraction and data encapsulation features.
- Example: Fruit mango
- In above example, mango is an object which is created from class Fruit.

3) Data Abstraction:
- To show only essential details without background details.

4) Data Encapsulation:
- The wrapping up of data and functions into a single unit is known as Data Encapsulation.

5) Inheritance:
- The process of creating new class by using the concept of old class is known as Inheritance.
- Newly created class is known as Derived class.

- Old class is known as Base class.

6) Polymorphism:
- Polymorphism is a greek word.
- Poly means 'many' and morphism means 'forms'.
- Ability to take more than one forms is known as Polymorphism.
- There are two different types of Polymorphism
-> Compile time Polymorphism
   => Function Overloading
   => Operator Overloading //this is not supported in java
-> Run time Polymorphism
   => Virtual Function

7) Dynamic Binding:
- The linking between calling function and called function is known as binding.
- But that linking is not known until the execution of program is known as dynamic binding.
- Example:

```
vjtech();   //calling function
void vjtech() //called function
{
  //body
}
```

8) Message Passing:
- In OOP, we can create set of objects that communicate with each other.
- I) Creating classes that define objects.
  II) Creating objects from the class definition.
  III) Establishing communication among objects.

==========
Benefits of OOP
==========
1) Using concept inheritance, we can achieve reusability.
2) Data hiding
3) Software complexity can be easily managed.
4) Object oriented system can be easily upgraded from small to large systems.
5) It is easy to partition work in the project based on objects.

==========
Application of OOP
==========
1) Real time systems.
2) Simulation and modeling
3) Object-oriented databases
4) Hypertext,hypermedia and experttext.
5) AI and expert systems.
6) Neaural networks and parallel programming.
7) Decision support and office automation System.
8) CIM/CAD System

Java History:

==========
- Java is a general purpose, object oriented programming language.
- It was developed by Sun MircoSystem of USA.
- James Gosling is owner of Java language.
- It was developed in year 1991.
- Initially, it was called as Oak (tree name)
- It's name got changed to Java in year 1995.

==========
Java Features:
==========
1) Compiled and Interpreted:

- Usually computer language is either compiled or interpreted.
- But java combines both these approaches thats via Java is called as two stage compilation process programming language.
- Java compiler takes java source file(.java) as input and generates byte file(.byte).
- Byte file is not a machine code and this file not exists physically in your machine.
- Byte code generated virtually and process virtually.
- Java Interpreter generates machine code from byte code.

2) Platform Independent and Portable:

- Java program can be easily moved from one computer to another computer,anywhere and anytime.
- It means, if we develop Java code on Windows machine then you can easily run that code on other operating systems like Linux,Unix,etc.
- To move java code from one machine to another machine is known as portability.

3) Object Oriented:

- Java is true object-oriented language.
- Almost everything in java is an object.
- All program code and data reside within objects and classes.
- Java is a colletion of rich set of predefined classes and packages, that we can use
in our programs by inheritance.

4) Robust & Secure:

- Java is robust language.
- It provides many safeguards to ensure reliable code.
- It has strict compile time and runtime checking for data types.
- It supports garbage collection feature that would solve memory management problem.
- It supports exception handling which help us to captures many errors.
- Java is more secure programming language which is used for programming on internet.

- Java systems not only verify all memory access but also ensure that no viruses are
communicated with an applet.

5) Distributed:

- Java is designed as distributed langauge for creating applications on networks.
- It has ability to share both data and programs.
- Java application can open and access remote objects on internet.
- This enables multiple programmers at multiple locations to collaborate and work together on a single object.

6) Simple, Small and Familiar:

- Java is a small and simple language.
- Many features of C & C++ which are not relible that was not added in Java.
- Java does not use pointers, preprocessor directive, goto statement and many others.
- Also not included multiple inheritance and operator overloading features.
- Familiarity is another important feature of Java.

7) Multi-threaded and Interactive:

- Thread is a light weight process because it takes small amount of memory space for their execution.
- When mutliple thread executes simultaneously then it is called as Multithreading.
- Java supports multithreaded programs.
- This means that we need not to wait for the application to finish one task before

beginning another.
- Due to this feature, we can create more interactive programs in java.

8) High Performance:

- Java performance is impressive for an interpreted language, mainly due to the use

of byte code.
- Java speed is more faster than C/C++ language.
- Java architecture is also designed to reduce overhead during the runtime.
- Due to multi-threading fetures Java program execution speed is increased.

9) Dynamic & Extensible:

- Java is dynamic language.
- Java is capable of dynamically linking in new class libraries,methods and objects.
- Java programs support functions written in othet languages such as c and C++.
- This facility enables the programmers to use the efficient functions in these languages.
- It is called as native functions/methods.

- Native methods are linked dynamically at runtime.

==========
Difference between Java and C
==========
1) Java is Object oriented programming language and C is Procedure oriented programming.
2) Java does not include c keyword sizeof and typedef.
3) Java does not contain data types struct and union.
4) Java does not define the data type modifiers keyword auto,extern,register,signed and
unsigned.
5) Java does not support pointer concept.
6) Java does not have preprocessor directive and thats via we don't use #define, #include
7) Java adds labeled break and continue statements.
8) Java adds new operators such as instanceof.
9) Java adds many required features of Object Oriented Programming Language.

==========
Difference between Java and C++
==========
1) C++ is object-oriented programming but Java is true object-oriented programming.
2) Java does not support operator overloading.
3) Java does not support pointer concept.
4) Java does not support multiple inheritance. But you can implement it using new feature called 'interface'.
5) Java does not support global variable.
6) Java does not support destructor function but we use finalize() method.
7) There is no header files in Java.
==========
Java Environment
==========
- Java environment includes development tools(JDK) and classes & method(JSL-API).
- JDK stands for Java development kit.
- JSL stands for Java Standard Library.
- JRE stands for Java Runtime Environment.

1) Java Development kit(JDK)

=> javac : Java compiler (javac filename.java)
=> java : Java Interpreter (java filename)
=> jdb : Java Debugger
=> appletviewer : for running java applets.
=> javah : for c header files.
=> javadoc : for creating HTML documents.
=> javap : Java disassembler(convert byte file to program)

Comments:
==========
- Comments part ignored by the compiler.
1) Single line comment
//This is single line comment
2) Multi-line comment
/*
This is multi-line comment
*/

Command line arguments:

- Give input to Java code.
- Example: java filename <list of input values>
                  java VJTech 100 200
- Program:
//Command line arguments
class CommandLineArgsDemo
{
        public static void main(String args[])
        {
                int a,b,c;
                a=Integer.parseInt(args[0]);
                b=Integer.parseInt(args[1]);
                c=a+b;
                System.out.println("Addition of two numbers="+c);
        }
}

Scanner Class:

Method   Description
-------   ------------
nextInt()   reads an int value from the user
nextFloat()   reads a float value form the user
nextBoolean()   reads a boolean value from the user
nextLine()   reads a line of text from the user
next()   reads a word from the user
nextByte()   reads a byte value from the user
nextDouble()   reads a double value from the user
nextShort()   reads a short value from the user
nextLong()   reads a long value from the user

Mathematical Functions:
========================
- java.lang package contain Math class.
- If you want to access methods of Math class then use below syntax:
  Math.MethodName();

Math.min(Variable1,Variable2) - find minimum value

```
Math.max(Variable1,Variable2) - find maximum value
Math.sqrt(VariableName)                 - find square root of given number
Math.pow(Variable1,Variable2) - return power of given number
Math.exp(VariableName)          - to calculate exponential value of given number
Math.round(Variable)                    - it return rounded value.
Math.abs(Variable)                      - It is used to find out absolute value.

- Example:
class MathMethods
{
        public static void main(String args[])
        {
                int m=12,n=12;
                System.out.println("The minimum Value = "+Math.min(m,n));
                System.out.println("The maximum Value = "+Math.max(m,n));
                System.out.println("Square root of 9  = "+Math.sqrt(9));
                System.out.println("Pow(2,3)          = "+Math.pow(2,4));
                System.out.println("exponential of 709.78222656 is
"+Math.exp(709.78222656));
                System.out.println("round(200.675)    = "+Math.round(200.675));
                System.out.println("round(200.675)    = "+Math.round(200.50));
                System.out.println("round(200.675)    = "+Math.round(200.20));
                System.out.println("Absolute Value    = "+Math.abs(-5944));
        }
}
/*OUTPUT
F:\Academic 2022\JavaBatch2022\UNI-I Official>java MathMethods
The minimum Value = 10
The maximum Value = 12
Square root of 9  = 3.0
Pow(2,3)          = 16.0
exponential of 709.78222656 is 1.7968190692375724E308
round(200.675)    = 201
round(200.675)    = 201
round(200.675)    = 200
Absolute Value    = 5944
*/

Data Types:
===========

Constants:

====================
Scope of Variables:
====================
- Scope of the variables is nothig but the life time of variables.
- Its scope is depend on where in the program that variables are declared.
- The area of the program where the variable is accessible is called as scope.
- There are three different types of variables present in java
```

1) Instance Variables:

- Instance variable is declared inside the class.
- Instance variables are created when the objects are instantiated.
- Instance variables allocate separate memory space when object is created.
- They take different values for each object.

2) Class Variables:

- Class variables are declared inside the class.
- They are the global to the class.
- It common between all objects.
- Only one memory location is created for each class variables.

3) Local Variables:

- Local Variables declared and used inside the functions.
- The variables which are declared inside the body of methods in known as local variables.
- They are not available outside the method.
- Local variables can be declared inside the body of methods which is starting from opening curly braces ({} and closing braces(}).

Example:

```
class Student
{
        int rollno;                                             //instance variable
        String name;                                    //instance variable
        float marks;                                    //instance variable
        static int college_code=1010;   //class variable
        void calc_marks()
        {
                        int total;              //local variable
        }

}
```

==========
Type casting/ Data Type conversion
==========
- The process of converting one data type to another data type is known as type casting.
- To change entity of one data type to another data type is known as data type conversion.
- Tyep casting occurs when we want to store value of one data type into variable of another type.
- This type casting is required while developing applications.
- If you store large data type value into small data type then it might be data loss.

- If you will store an int value into byte variable then this will be illegal operation.
- To avoid data loss, you should store smaller data type value into larger data type
variable.
- Conversion Table:

From	To
byte	short,char,int,long,float,double
short	int,long,float,double
char	int,long,float,double
int	long,float,double
long	float,double
float	double

- There are two types of casting:
1) Implicit Type casting
- The type casting which is done by the system is known as Implicit type casting.
- Example:

```
//Implicit Type casting
class ImplicitTypeCastingDemo
{
		public static void main(String args[])
		{
				int a=70;
				float b;
				b=a;
				System.out.println("Value of int variable a="+a);
				System.out.println("Value of float variable b="+b);
		}
}
```

2) Explicit Type Casting
- The type casting which is done by the programmer is known as Explicit type casting.
- Syntax:

```
datatype VariableName1=(datatype)VariableName2;
```

- Example:

```
//Explicit Type casting
class ExplicitTypeCastingDemo
{
		public static void main(String args[])
		{
				int a=70;
				float b;
				b=(float)a;
				System.out.println("Value of int variable a="+a);
				System.out.println("Value of float variable b="+b);
		}
}
```

```
=======================
Standard Default Values
=======================
- Every variable has default value in JAVA.
- If variable is not initialized then java provides default value to that varible
automatically.

Type Of Variables               Default Value
---------------------------------------
byte                                    zero(0)
short                                   zero(0)
int                                             zero(0)
long                                    zero(0L)
float                                   0.0f
double                                  0.0d
char                                    Null Character
boolean                                 false

==============================
Operators and Expression
==============================
- Operator is a symbol which indicate operation to be perform.
- Operands is a variable on which we can perform operation.
- The proper arrangement of operators and operands is known as Expression.
- Following are the classification of operators in JAVA:
1) Arithmetic Operators(+,-,*,/,%)
2) Relational Operators(<,>,<=,>=,==,!=)
3) Logical Operators(&&,||,!)
4) Assignment Operators(=)
5) Increment and decrement Operators(++,--)
6) Conditional Operator(?:) condition?expression1:expression2;
7) Bitwise Operator(&,|,^,<<,>>,~)
8) Special operators(instanceof, dot)

Instanceof Operator:
--------------------
- This operator return true if the object on the left side is an instance of the
class
given on the right side.
- Syntax:

        if(object instanceof ClassName)
        {
                //body
        }

- Example:
//instanceof operator
import java.util.*;
class InstanceOfOpDemo
```

```
{
	public static void main(String args[])
	{
		Scanner sc=new Scanner(System.in);

		if(sc instanceof Scanner)
		{
				System.out.println("sc is an object of Scanner
class");
		}
	}
}

==============================
***Decision Making Statements:
==============================
1) simple if statement:
----------------------
- if predefined keyword.
- Syntax:
	if(condition)
	{
		//body of if.
	}

- Example:
	//Write a Java program to check whether two numbers are same or not.
	import java.util.*;
	class IfStatement
	{
			public static void main(String args[])
			{
					int a,b;

					Scanner sc=new Scanner(System.in);
					System.out.println("Enter Two Integer
Numbers:");
					a=sc.nextInt();
					b=sc.nextInt();

					if(a==b)
					{
						System.out.println("Both number are
equal!!!");
					}
			}
	}
	/*
	Enter Two Integer Numbers:
	100
```

```
        100
        Both number are equal!!!
        */

2) if-else Statement:
======================
- if and else both are predefined keywords.
- syntax:
                        if(condition)
                        {
                                //body of if
                        }
                        else
                        {
                                //body of else
                        }
- if condition is true then program controller executes if body otherwise executes
else part.
- Example:
        //Write a Java program to check whether entered number is even or ODD
        import java.util.*;
        class IfElseStatement
        {
                        public static void main(String args[])
                        {
                                        int no;
                                        Scanner sc=new Scanner(System.in);
                                        System.out.println("Enter Any Integer
Number:");
                                        no=sc.nextInt();

                                        if(no%2==0)
                                        {
                                                System.out.println("Number is
EVEN");
                                        }
                                        else
                                        {
                                                System.out.println("Number is
ODD");
                                        }
                        }
        }
        /*
        Enter Any Integer Number:
        15
        Number is ODD
        */

3) Nested if-else statement:
```

```
-----------------------------
- One if-else within another if is known as nested if-else statement.
- Syntax:
                        if(condition-1)
                        {
                                if(condition-2)
                                {
                                                //body of if
                                }
                                else
                                {
                                                //body of else
                                }
                        }
                        else
                        {
                                //body of else
                        }
- Example:
        //write a Java program to check whether number is positive or negative
        import java.util.*;
        class NestedIfElseStmt
        {
                public static void main(String args[])
                {
                        int no;
                        Scanner sc=new Scanner(System.in);

                        System.out.println("Enter any Integer Number:");
                        no=sc.nextInt();
                        if(no!=0)
                        {
                                if(no>0)
                                {
                                                System.out.println("Number is
Positive!!!");
                                }
                                else
                                {
                                                System.out.println("Number is
Negative!!!");
                                }
                        }
                        else
                        {
                                System.out.println("Zero is neigher Positive nor
Negative");
                        }

                }
```

```
        }
        /*
        F:\Academic 2022\JavaBatch2022\UNI-I Official>java NestedIfElseStmt
        Enter any Integer Number:
        143
        Number is Positive!!!

        F:\Academic 2022\JavaBatch2022\UNI-I Official>java NestedIfElseStmt
        Enter any Integer Number:
        -23
        Number is Negative!!!

        F:\Academic 2022\JavaBatch2022\UNI-I Official>java NestedIfElseStmt
        Enter any Integer Number:
        0
        Zero is neigher Positive nor Negative
        */
```

4) else-if ladder
==========
- Suppose, we have multiple conditions but in which only one condition will get true then we can use else if ladder.
- Syntax:

```
                        if(condition-1)
                        {
                                //block of statements
                        }
                        else if(condition-2)
                        {
                                //block of statements
                        }
                        else if(condition-N)
                        {
                                //block of statements
                        }
                        else
                        {
                                //block of statements
                        }
```

- Example:

```
        /*Write a Java program to generate student mark grade on the basis of following conditions.
        marks>=75 - Distinction
        marks>=60 - First Class
        marks>=40 - Pass
        marks<40  - Fail
        */
        import java.util.*;
        class ElseIfLadder
```

```
	{
		public static void main(String args[])
		{
			int marks;
			Scanner sc=new Scanner(System.in);

			System.out.println("Enter Your Marks:");
			marks=sc.nextInt();

			if(marks>=75)
			{
					System.out.println("Congratulations...You
got Distinction");
			}
			else if(marks>=60)
			{
					System.out.println("Congratulations...You
got First Class");
			}
			else if(marks>=40)
			{
					System.out.println("Congratulations...You
are Pass Only");
			}
			else
			{
					System.out.println("You are Fail!!!");

			}
		}
	}
	/*
	F:\Academic 2022\JavaBatch2022\UNI-I Official>java ElseIfLadder
	Enter Your Marks:
	88
	Congratulations...You got Distinction

	F:\Academic 2022\JavaBatch2022\UNI-I Official>java ElseIfLadder
	Enter Your Marks:
	65
	Congratulations...You got First Class

	F:\Academic 2022\JavaBatch2022\UNI-I Official>java ElseIfLadder
	Enter Your Marks:
	58
	Congratulations...You are Pass Only

	F:\Academic 2022\JavaBatch2022\UNI-I Official>java ElseIfLadder
	Enter Your Marks:
	31
```

```
        You are Fail!!!
        */

switch case statement:
======================
- switch, case, break and defualt keywords.
- Syntax:
                        switch(expression/value)
                        {
                                case value-1: //block of statements
                                                        break;
                                case value-2: //block of statements
                                                        break;
                                case value-N: //block of statements
                                                        break;
                                default: //block of statements
                        }

Looping Statements:
=======================
1) for loop
2) while loop
3) do while loop
4) Enhanced for loop

For Loop:
---------------
- Syntax:
                for(initialization;condition;incre/decre)
                {
                        //body of for loop
                }
- Example:
        //for loop
        import java.util.*;
        class forloopDemo
        {
                public static void main(String args[])
                {
                        int i;
                        for(i=1;i<=5;i++)
                        {
                                System.out.println("VJTech Academy");
                        }
                }
        }
        /*
        VJTech Academy
        VJTech Academy
        VJTech Academy
```

```
        VJTech Academy
        VJTech Academy
        */

while loop:
==============
- while is a predefined keyword
- Syntax:
                while(Condition)
                {
                        //body of while loop
                }
- Example:
        //for loop
        import java.util.*;
        class whileloopDemo
        {
                public static void main(String args[])
                {
                        int i=1;
                        while(i<=5)
                        {
                                System.out.println("VJTech Academy");
                                i++;
                        }
                }
        }
        /*
        VJTech Academy
        VJTech Academy
        VJTech Academy
        VJTech Academy
        VJTech Academy
        */

do-while loop:
==================
- do & while both are predefined keywords.
- Syntax:
                do
                {
                        //body
                }while(condition);
- Example

Enhanced for loop/For each loop:
================================
- It is also called as for each loop.
- Using this loop, we can easily retrieve the value of array without using indexes.
- Using for each loop, we can easily iterate over the array.
```

- Syntax:

```
for(DataType VariableName:Expression)
{
        //statements
}
```

- Example:

```
//for each loop
class ForEachLoopDemo
{
        public static void main(String args[])
        {
                int num[]={10,20,30,40,50};

                System.out.println("Your Array Elements are:");

                for(int x:num)
                {
                        System.out.println("Value of x :"+x);
                }
        }
}
/*
Your Array Elements are:
Value of x :10
Value of x :20
Value of x :30
Value of x :40
Value of x :50
*/
```

```
**********************************************
*  Class Name : VJTech Academy , Maharashtra     *
*  Author Name: Vishal Jadhav Sir                *
*  Mobile No  : 9730087674                       *
**********************************************
====================================
UNIT-II :Classes, Objects and Methods
====================================

***Defining the Class:
======================
- Class is a collection of similiar types of objects.
- Class is a user defined data type.
- Class contains instance variables and methods.
- Class shows data abstraction and data encapsulation properties.
- Syntax:
        class Class_Name
        {
                datatype instance_variable1;
                datatype instance_variable2;
                datatype instance_variableN;
                returntype Method_Name1(parameter_list)
                {
                        //body of method
                }
                returntype Method_Name2(parameter_list)
                {
                        //body of method
                }
                returntype Method_NameN(parameter_list)
                {
                        //body of method
                }
        }
- Example:
        class Addition
        {
                int a,b,c;
                void getdata()
                {
                        a=100;
                        b=50;
                }
                void display()
                {
                        c=a+b;
                        System.out.println("Addition="+c);
                }
        }
```

Note:
-----
1) No semicolon after the closing curly bracket.
2) ClassName is valid java identifier
3) The instance variable and methods defined inside the class is known as member of

   the class.
4) Class declaration only creates template, it does not create an actual object.
5) Memory should not be allocated for the data members of the class.
6) Method definition should present inside the class only.

***Creating an Objects:
======================
- An object is instance of the class.
- When object is created then memory will be allocated for the instance variables
of the class.
- We can create N no of objects from the class.
- Objects are created using new operator.
- The new operator creates an objects of the class and return reference to the
object
  created.
1) Declaration of Object:
   Syntax : ClassName ObjectName;
   Example: Addition a1;

2) Instantiate of Object:
   Syntax : ObjectName=new ClassName(Parameter_list);
   Example: a1=new Addition();

- Combining the above two steps:
  Syntax : ClassName ObjectName=new ClassName(Parameter_list);
  Example: Addition a1=new Addition();

***Accessing Class Members:
==========================
- Object contain data members and member function.
- So we can access them by using object name and dot operator.
- Syntax:
        ObjectName.Variable_Name=value;
        ObjectName.Method_Name(Parameter_list);
- Example:
        a1.a=100;
        a1.b=200;
        a1.getdata();
        a1.display();

***Array of objects:
====================
- Array of object is the collection of objects of same class.
- Instead of creating mutiple objects, it would be better to create array of

objects.
- Array index should begin with 0 and end with size-1.
- Object Name is same but its indexes are different.
- Syntax:

```
ClassName ObjectName[]=new ClassName[SIZE];
```

- Example:

```
VJTech v[]=new VJTech[5];
for(int i=0;i<5;i++)
{
v[i]=new VJTech();//Assigning object to individual reference in the array.
}
```

==========
Constructors
==========
- Constructor is a special member function of the class.
- It is used to initialize the data members of the objects.
- There is no any return type for the constructor.
- Constructor name and class name both are same.
- Constructor automatically called when object is created.
- Constructor is used for creation of an object.
- Suppose, in our class we have not defined constructor then system will supply the default constructor for creation of an objects.
- There are three different types of the constructor:

1) Default constructor
2) Parameterized constructor
3) Copy constructor

1) Default constructor:

- When constructor does not takes any parameters then it is called as default constructor.
- Syntax:

```
class ClassName
{
        ClassName()
        {
                //body of constructor.
        }
}
```

2) Parameterized constructor:

- When constructor takes any parameters then it is called as Parameterized constructor.
- Syntax:

```
class ClassName
{
        ClassName(parameter_list)
        {
                //body of constructor.
        }
```

```
                }
```

2) Copy constructor:

- To initialize data members of the object, we are passing another object as argument is called as copy constructor.
- When constructor takes reference of its class as paramater then it is called as copy constructor.
- Syntax:

```
                class ClassName
                {
                        ClassName(ClassName ObjectName)
                        {
                                //body of constructor.
                        }
                }
```

- Example:

```
//copy constructor
class Item
{
        int x;
        Item()
        {
                x=100;
        }
        Item(Item m)
        {
                x=m.x;
        }
        void display()
        {
                System.out.println("Value of X : "+x);
        }
        public static void main(String args[])
        {
                Item i1=new Item();
                Item i2=new Item(i1);
                i1.display();
                i2.display();
        }
}
```

***Constructor overloading:
==========
- Constructor names are same but its arguments are different.
- Example:

```
//constructor overloading
class Room
{
        float length;
```

```
        float width;
        Room(float x)
        {
                length=x;
                width=20.50f;
        }
        Room(float m,float n)
        {
                length=m;
                width=n;
        }
        void display()
        {
                float room_area;
                room_area=(length*width);
                System.out.println("Area of Room : "+room_area);
        }
        public static void main(String args[])
        {
                Room r1=new Room(10.10f);
                Room r2=new Room(50.50f,23.50f);

                r1.display();
                r2.display();
        }
}
```

==========
Static Members
==========

- Static members can be data and methods.
- When we create objects of the class then separate memory allocated for each data members of the class.
- But sometime, there could be situation where we want to keep one variable common between all objects.
- In this case, we can make that variable as static.
- When we make variable as static then only one copy of that variable created in computer memory and all objects share it commonaly.
- Static members comes under the scope of class.
- We can access static members using classname and dot operator.
- Inside the body of static member function, we can access only other static data members.
- There is no need to define static data member outside the class.
- By default static variable contain zero value.
- Example:

```
//static data member and static member function
class StaticDemo
{
        int no;
        static int count;               //static variable
        void getdata(int x)
```

```
        {
                no=x;
                count++;
        }
        void display_no()
        {
                System.out.println("Value of no="+no);
        }
        static void display_count() //static method
        {
                System.out.println("Value of count="+count);
        }
        public static void main(String args[])
        {
                StaticDemo s1=new StaticDemo();
                StaticDemo s2=new StaticDemo();
                StaticDemo s3=new StaticDemo();

                s1.getdata(100);
                s2.getdata(200);
                s3.getdata(300);

                System.out.println("Object s1 :");
                s1.display_no();
                StaticDemo.display_count();

                System.out.println("Object s2 :");
                s2.display_no();
                StaticDemo.display_count();

                System.out.println("Object s3 :");
                s3.display_no();
                StaticDemo.display_count();
        }
}

OUTPUT:
---------
Object s1 :
Value of no=100
Value of count=3
Object s2 :
Value of no=200
Value of count=3
Object s3 :
Value of no=300
Value of count=3
================================================
Visibility Control/Access Specifiers Parameter:
================================================
```

- Java provides four types of visibility control.
1) public
2) private
3) protected
4) Friendly Access

public:

- Any variables and methods declared as public, it can be accessible outside the class.

private:

- Those members are declared as private, it can accessible within the class in which they are declared.
- It cannot be inherited in its subclass.

protected:

- Those members are declared as protected, it can accessible in same class and its immediate sub-class.

Friendly Access:

- In the situation where no access modifier is specified then by default all members considered as friendly access level.
- There is basic difference between public and friendly access is that public members accessible anywhere but friendly access member available in same package not outside the package.

==========
Arrays
==========
- Normally, one variable can store one value at a time.
- But sometimes, we need to store multiple values then creation of multiple variables is not a better solution.
- In this case, we can create array variable.
- Array variable name is same but it will store multiple values.
- Array is a collection of similiar types of elements.
- Array index should begin with Zero and end with SIZE-1.
- Array elements are stored in continues memory location.
- There are three different types of array:
1) One dimensional array
2) Two dimensional array
3) Multi-dimensional array

- Creating an Array involes following steps:
I) Declaration of Array:
- Syntax:
```
        form-1  datatype ArrayName[];
```

```
        form-2  datatype [] ArrayName;
- Example:
                        int marks[];
                        float average[];
                        int [] sum;

II) Creation of Array:
- After declaration of array, we need to create it in the memory. Java allows us to
create array using new operator only.
- Syntax:
                        ArrayName=new datatype[SIZE];
- Example:
                        marks=new int[5];
                        average=new float[10];

- Combining step-I and step-II:
Syntax: datatype ArrayName[]=new datatype[SIZE];
Example: int marks[]=new int[5];

Array Initialization:
---------------------
- In this step, we will put values into the array.
- This process is known as initialization.
- We can initialize array elements using index number and subscripts.
- Syntax:
                        ArrayName[index]=value
- Example:
                        marks[0]=99;
                        marks[1]=78;
                        marks[2]=65;
- We can also initialize array elements automatically at the time of declaration.
- Syntax:
                        datatype ArrayName[]={list of values};
- Example:  int number[]={10,20,30,40,50};

Array Length:
-------------
- Java provides predefined method length to calculate length of array.
- Example:
                int a[]={10,20,30};
                int len=a.length;
                len=3;
- Example-1:
class OneDArray
{
        public static void main(String args[])
        {
                int a[]=new int[5];
                a[0]=10;
                a[1]=20;
```

```
                a[2]=30;
                a[3]=40;
                a[4]=50;
                for(int i=0;i<a.length;i++)
                {
                   System.out.println("Element present at index "+i+" is "+a[i]);
                }
        }
}
/*
Element present at index 0 is 10
Element present at index 1 is 20
Element present at index 2 is 30
Element present at index 3 is 40
Element present at index 4 is 50
*/
```

- Example-2:

```
import java.util.*;
class OneDArray1
{
        public static void main(String args[])
        {
                int a[]=new int[5];
                int i;
                Scanner sc=new Scanner(System.in);
                System.out.println("Enter Five Array Elements:");
                for(i=0;i<5;i++)
                {
                   a[i]=sc.nextInt();
                }
                System.out.print("Array Elements are : ");
                for(i=0;i<5;i++)
                {
                   System.out.print(a[i]+" ");
                }
        }
}
/*
Enter Five Array Elements:
100
200
300
400
500
Array Elements are : 100 200 300 400 500
*/
```

- Example-3:

```
//one dimensional Array : reverse the elements of array
```

```
import java.util.*;
class OneDArray2
{
        public static void main(String args[])
        {
                int a[]=new int[5];
                int i;
                Scanner sc=new Scanner(System.in);
                System.out.println("Enter Five Array Elements:");
                for(i=0;i<5;i++)
                {
                   a[i]=sc.nextInt();
                }
                System.out.print("Display Array Elements in Reverse Order: ");
                for(i=a.length-1;i>=0;i--)
                {
                   System.out.print(a[i]+" ");
                }
        }
}
/*
Enter Five Array Elements:
100
200
300
400
500
Display Array Elements in Reverse Order: 500 400 300 200 100
*/
```

```
======================
Two Dimensional Array
======================
- Two Dimensional array is used to maintain elements in rows and column format.
- We can represent data in tabular format.
- Rows index should begin with 0 and end with size-1.
- Columns index should begin with 0 and end with size-1.
- Creating an Array involes following steps:
I) Declaration of Array:
- Syntax:
        form-1  datatype ArrayName[][];
        form-2  datatype [][] ArrayName;
- Example:
                        int marks[][];
                        float average[][];
                        int [][] sum;

II) Creation of Array:
- After declaration of array, we need to create it in the memory. Java allows us to
create array using new operator only.
```

- Syntax:
```
                        ArrayName=new datatype[ROWS][COLUMNS];
```
- Example:
```
                        marks=new int[3][3];
                        average=new float[10][10];
```

- Combining step-I and step-II:

Syntax: datatype ArrayName[][]=new datatype[ROWS][COLUMNS];

Example: int marks[][]=new int[3][3];

Array Initialization:

- In this step, we will put values into the array.
- This process is known as initialization.
- We can initialize array elements using index number and subscripts.
- Syntax:
```
                        ArrayName[Row-Index][Column-Index]=value;
```
- Example:
```
                        int a[][]=new int[3][3];
                        a[0][0]=10;
                        a[0][1]=20;
                        a[0][2]=30;
                        a[1][0]=40;
                        a[1][1]=50;
                        a[1][2]=60;
                        a[2][0]=70;
                        a[2][1]=80;
                        a[2][2]=90;
```

- We can also initialize array elements automatically at the time of declaration.
- Syntax:
```
                        datatype ArrayName[][]={list of values};
```
- Example:
```
int number[][]={{10,20,30},
                                                    {40,50,60},
                                                    {70,80,90}
                                                };
```

- Program:
```
import java.util.*;
class TwoDArray
{
        public static void main(String args[])
        {
                int a[][]=new int[3][3];
                int i,j;
                Scanner sc=new Scanner(System.in);
                System.out.println("Enter 3*3 Array Elements:");
                for(i=0;i<3;i++)
                {
                        for(j=0;j<3;j++)
```

```
                        {
                                a[i][j]=sc.nextInt();

                        }
                }
                System.out.println("Your 3*3 Array Elements: ");
                for(i=0;i<3;i++)
                {
                        for(j=0;j<3;j++)
                        {
                                System.out.print(a[i][j]+"  ");
                        }
                        System.out.println();
                }
        }
}
/*
Enter 3*3 Array Elements:
10
20
30
40
50
60
70
80
90
Your 3*3 Array Elements:
10  20  30
40  50  60
70  80  90
*/
=================
***Vectors***
=================
- Vector is an extensible array.
- Vector is a collection of objects and it can be retrived by using index number.
- Array is a collection of similiar types of elements but its size is fixed.
- But vector is a collection of objects but its size is not fixed.
- Vector class provides array of variable size.
- The main difference between vector and array : Vector automatically grow when
they run out of space.
- Vectors class provides extra method for adding and removing elements.
- The class is used to create dynamic array known as Vector that can holds objects
of any type and any numbers.
- Vector is a predefined class which is present under java.util package.

- Vectors are created like array as follow:
1) Declaration of Vector without size.
   Vector VectorName=new Vector():
```

2) Declaration of Vector with size.
   Vector VectorName=new Vector(10):

Advantages of Vector over arrays:
---
1) It is convenient to the vector to store objects.
2) Vector is used to store multiple objects and its size not fixed.
3) Objects can be added and deleted from the vector whenever required.

DisAdvantages of Vector
---
1) It can not directly store simple data type only objects can be stored.
2) To store simple data type value in the vector, they must be converted into objects.
3) Wrapper classes are required for above conversion.

Important Vector Methods:

Vector v1=new Vector();

1) v1.addElement(item) - Adds the item to the vector at the end.
2) v1.elementAt(10)   - Gives the name of 10th object.
3) v1.size()   - Gives the number of objects present.
4) v1.removeElement(item) - Removes the specified item from the vector.
5) v1.removeElementAt(n) - Removes the item stored in nth position.
6) v1.removeAllElements() - Removes all the elements in the list.
7) v1.copyInto(array)   - Copies all items from vector to array.
8) v1.insertElementAt(item,n) - Insert the item at nth position.

Program-1:

```
/*Write a program to create vector with six elements (10,30,60,70,80,100). Removes element 3rd and 4th position. Insert new element at 3rd position. Display the original and current size of vector
*/
import java.util.*;
class VectorDemo1
{
        public static void main(String args[])
        {
                Vector v1=new Vector();
                v1.addElement(new Integer(10));
                v1.addElement(new Integer(30));
                v1.addElement(new Integer(60));
                v1.addElement(new Integer(70));
                v1.addElement(new Integer(80));
                v1.addElement(new Integer(100));
                System.out.println("Initial Vector Size = "+v1.size());
                v1.removeElementAt(3);
```

```
                v1.removeElementAt(4);
                v1.insertElementAt(new Integer(150),3);
                System.out.println("Final Vector Size = "+v1.size());

        }
}
/*
Initial Vector Size = 6
Final Vector Size = 5
*/
------------
Program-2:
-----------
import java.util.*;
class VectorDemo2
{
        public static void main(String args[])
        {
                Vector v1=new Vector();
                for(int i=0;i<args.length;i++)
                {
                        v1.addElement(args[i]);
                }
                System.out.println("Vector of Size = "+v1.size());
                System.out.println("Your Vector Elements:");
                for(int i=0;i<v1.size();i++)
                {
                        System.out.println(v1.elementAt(i));
                }
        }
}
/*
F:\Academic 2022\JavaBatch2022\UNIT-II>java VectorDemo2 10 20 30 40 50
Vector of Size = 5
Your Vector Elements:
10
20
30
40
50
*/
================
Wrapper Classes
================
- As you already aware about, in java some concepts only works on objects.
- Let take one example - Vector : Vector cannot handle primitive data types like
int,float,long,char and double.
- Using Wrapper classes, we can convert primitive data type into objects.
- In Java different wrapper clases are given which is present under java.lang
package.
```

- Following table shows wrapper classes for converting simple types:

Simple Type	Wrapper Class
1) boolean	Boolean
2) char	Character
3) float	Float
4) double	Double
5) int	Integer
6) long	Long

- The wrapper classes have number of unique method for handling primitive data type and objects.

I)Converting primitive numbers to object numbers using constructor methods

Constructor Calling	Conversion Action
1)Integer x=new Integer(i)	Primitive integer to Integer object
2)Float y=new Float(f)	Primitive float to Float object.
3)Double z=new Double(d)	Primitive double to Double object.
4)Long w=new Long(l)	Primitive long to Long object.
5)Character v=new Character(c)	Primitive char to Character object.

Note: i, f, d, l, c are primitive data type values which denoting int,float,double,long and char data types.

Example:

```
int i=100;                                          //Primitive integer i
Integer x=new Integer(i):   // Integer Object x

float f=100.10;                          //Primitive float f
Float y=new Float(f):      // Float Object y
```

II)Converting Object numbers to primitive numbers using typeValue() method

Method Calling	Conversion Action
1)int i=x.intValue();	Object to primitive int
2)float f=y.floatValue();	Object to primitive float
3)double d=z.doubleValue();	Object to primitive double
4)long l=w.longValue();	Object to primitive long
5)char c=v.charValue();	Object to primitive char

III)Converting numbers to string using toString() method

Method Calling	Conversion Action
1) str=Integer.toString(i)	Primitive Integer to String

```
2) str=Float.toString(f)                          Primitive float to String
3) str=Double.toString(d)                         Primitive double to String
4) str=Long.toString(l)                         Primitive Long to String

IV)Converting string objects to numeric objects using ValueOf() method
        Method Calling                          Conversion Action
     --------------------------          ------------------
1) DoubleVal=Double.ValueOf(str);     Converting string to Double Object
2) floatVal=Float.ValueOf(str);       Converting string to Float Object
3) intVal=Integer.ValueOf(str);       Converting string to Integer Object.
4) longVal=Long.ValueOf(str);             Converting string to Long object

V)Converting numeric string to primitive numbers using parsing methods
        Method Calling                          Conversion Action
     --------------------------          ------------------
1) int i=Integer.parseInt(str)        Converts string to primitive integer
2) long l=Long.parseLong(str)         Converts string to primitive long
3) float f=Float.parseFloat(str)      Converts string to primitive float
4) double d=Double.parseDouble(str)   Converts string to primitive double.

Program:
---------
class WrapperClassDemo
{
        public static void main(String args[])
        {
                int i=100;
                Integer ii=new Integer(i);
                System.out.println("Primitive Integer Value = "+i);
                System.out.println("Object Integer Value = "+ii);
        }
}
/*
Primitive Integer Value = 100
Object Integer Value = 100
*/

========================
Strings:
========================
- String is a sequence of characters.
- Collection of characters is known as String.
- String is a predefined class which is present under java.lang package.
- String should be represented by using double quotation.
- The easiest way to represent string in java is by using character array.
- Example:
                        char name[]=new char[4];
                        name[0]='J';
                        name[1]='a';
                        name[2]='v';
```

name[3]='a';
- By using above approach, we require lot of overhead to manage the string.
- For example, copy one character array into another array is difficult due to size issues.
- In java, we have one good approach to manage string i.e creation of string class object.
- This class provides lots of good methods to manipulate string.
- Syntax:

```
String strName="Value";
OR
String strName=new String("String");
```

- Example:

```
String firstName="Vishal";
String firstName=new String("Vishal");
```

- In java, we use + operator as concatenation operator.

String Arrays:

- We can also create and use arrays that contain strings.
- Example:

```
String x[]=new String[5];
```

- In above example, we can see string name is x and its size is 5. It means we can hold five string constants.

String Methods:

- String class provides different methods.

Method Names	Working
1) s1.toLowerCase();	convert the string s1 to lowercase
2) s1.toUpperCase();	convert the string s1 to Uppercase
3) s1.replace('x','y');	Replace all x with y in given string s1.
4) s1.trim();	Remove the spaces present at begin & end
5) s1.equals(s2)	Return true if s1 is equal to s2
6) s1.equalsIgnoreCase(s2)	Return true if s1==s2. It ignore the case.
7) s1.length()	Gives the length of s1.
8) s1.charAt(n)	Gives the nth character of s1.
9) s1.compareTo(s2)	s1==s2 : 0,s1>s2 : +ve value,s1<s2 : -ve value
10) s1.concat(s2)	Concatenates s1 and s2.
11) s1.substring(n)	Gives the substring starting from nth character.
12) s1.substring(n,m)	Gives substring starting from n & upto m characters.
13) p.toString()	Create string representation of object p.
14) s1.indexof('x')	Return index of x character in the string s1.
15) s1.indexof('x',n)	Return index of x which is occurred after nth character
16) String.valueof(p)	It will create string object of the parameter p.

- Example:

```
//String class
class StringDemo
{
        public static void main(String args[])
        {
                String str=new String("VJTech");

                System.out.println("Value of str ="+str);
                System.out.println("Length of str ="+str.length());
                System.out.println("To Lower Case ="+str.toLowerCase());
                System.out.println("To Upper Case ="+str.toUpperCase());
                System.out.println("Character present at 2 index ="+str.charAt(2));

                System.out.println("Concatenation="+str.concat("Academy"));
                System.out.println("Index of e character ="+str.indexOf('e'));
                System.out.println("Equals method ="+str.equals("VJTech"));
                System.out.println("CompareTo method ="+str.compareTo("VJTech"));

        }
}
/*
Value of str =VJTech
Length of str =6
To Lower Case =vjtech
To Upper Case =VJTECH
Character present at 2 index =T
Concatenation=VJTechAcademy
Index of e character =3
Equals method =true
CompareTo method =0
*/
```

StringBuffer class:

==========

- StringBuffer is a peer class of String.
- String class creates fixed length of string.
- StringBuffer class creates flexible length of string.
- We can insert characters and substring in the middle of the string.
- We can append another string to the end.
- Some methods of String class also supported in StringBuffer class.
- Commonly used StringBuffer methods:

1) s1.setCharAt(n,'x') - modifies the nth character to x.
2) s1.append(s2) - Appends the string s2 to s1 at the end.
3) s1.insert(n,s2) - Inserts the string s2 at the position n of the string s1.
4) s1.setLength(n) - Sets the length of string s1.
5) s1.length() - Gives the length of s1.
6) s1.charAt(n) - Gives the nth character of s1.

```
7) s1.equals(s2)         - Return true if s1 is equal to s2
- Example:
//StringBuffer class

class StringBufferDemo
{
        public static void main(String args[])
        {
                StringBuffer str=new StringBuffer("VJTech");

                System.out.println("Original String :"+str);
                System.out.println("Length of String :"+str.length());
                for(int i=0;i<str.length();i++)
                {
                        System.out.println("Character at position "+i+" is
"+str.charAt(i));
                }
                str.setCharAt(3,'T');
                System.out.println("Modified String :"+str);
                str.append("Academy");
                System.out.println("Appended String :"+str)     ;
        }
}
/*
Original String :VJTech
Length of String :6
Character at position 0 is V
Character at position 1 is J
Character at position 2 is T
Character at position 3 is e
Character at position 4 is c
Character at position 5 is h
Modified String :VJTTch
Appended String :VJTTchAcademy
*/
```

❖ **Difference between Array and vector:**

Array	Vector
Array size cannot be changed	A Vector is a dynamic array, whose size can be increased
Array is not a class	A Vector is a class
The Array can store similar type of values	Vectors can store any type of objects
Array is not synchronized	Vector is synchronized
Methods are not provided for adding and removing the element from the array	Methods are provided for adding and removing the element from the Vector.
The size of array needs to be declared in advance.	No need to declare the size of the vector. You may give its size & you may not.
The Array can store primitive data types	Vector can store only object
Once declared array can't grow in size	Vector can always grow in size if you start adding more element to it than your declared size

❖ **Difference between String and StringBuffer:**

String	StringBuffer
The length of the String object is fixed.	The length of the StringBuffer can be increased.
String object is immutable.	StringBuffer object is mutable
String object is slower in performance	StringBuffer object is faster..
Consumes more memory.	Consumes less memory.
String class overrides the equals() method of Object class. So you can compare the contents of two strings by equals() method.	StringBuffer class doesn't override the equals() method of Object class.
String objects are stored in a constant pool	StringBuffer objects are stored in heap memory.
String objects provides less functionality to the strings as compared to the class StringBuffer	StringBuffer objects provide more functionality to the strings as compared to the class String.

****
* Class Name : VJTech Academy , Maharashtra   *
* Author Name: Vishal Jadhav Sir   *
* Mobile No : 9730087674   *
****

==========
UNIT-III : Inheritance
==========

***Inheritance:

- The process of creating new class from old class is known as Inheritance.
- The mechanism of acquiring the properties of old class into the new class class is known as Inheritance.
- The newly created class is known as Subclass/child/derived class.
- Old class is known as Super class/Parent class/Base class.
- Inheritance will help us to achieve reusability feature.
- Because of Inheritance, our development time will get save and it will also impact on the project cost.

- Types of Inheritance:

1) Single Inheritance
2) Multi-level Inheritance
3) Multiple Inheritance
4) Hierarchical Inheritance
5) Hybrid Inheritance

- We use following syntax for creating the new class from old class.

```
class DerivedClassName extends BaseClassName
{
        //body of Derived Class
}
```

1) Single Inheritance:

- This is one of the types of inheritance.
- To create new class from only one base class is known as single inheritance.
- Syntax:

```
class DerivedClassName extends BaseClassName
{
        //body of Derived Class
}
```

- Program:

```
//Single Inheritance
import java.util.*;
class Student
{
        int rollno;
        String name;
```

```
        void get_stud_info()
        {
                Scanner sc=new Scanner(System.in);
                System.out.println("Enter Student Roll No:");
                rollno=sc.nextInt();
                System.out.println("Enter Student Name:");
                name=sc.next();
        }
        void disp_stud_info()
        {
                System.out.println("Student ROll No:"+rollno);
                System.out.println("Student Name:"+name);
        }
}
class Test extends Student
{
        int marks1,marks2;
        void get_stud_marks()
        {
                Scanner sc=new Scanner(System.in);
                System.out.println("Enter Student Test-1 Marks:");
                marks1=sc.nextInt();
                System.out.println("Enter Student Test-2 Marks:");
                marks2=sc.nextInt();
        }
        void disp_stud_marks()
        {
                System.out.println("Test-1 Marks:"+marks1);
                System.out.println("Test-2 Marks:"+marks2);
        }
}

class SingleInheritanceDemo
{
        public static void main(String args[])
        {
                Test t1=new Test();
                t1.get_stud_info();
                t1.get_stud_marks();
                System.out.println("*******STUDENT INFORMATION SYSTEM********");
                t1.disp_stud_info();
                t1.disp_stud_marks();
        }
}
/*
Enter Student Roll No:
1010
Enter Student Name:
James
Enter Student Test-1 Marks:
```

```
89
Enter Student Test-2 Marks:
78
*******STUDENT INFORMATION SYSTEM********
Student ROll No:1010
Student Name:James
Test-1 Marks:89
Test-2 Marks:78
*/
```

2) Multi-level Inheritance:

- The mechanism of deriving the class from another derived class is known as multi-level inheritance.
- To create new class from another derived class is known as multi-level inheritance.
- It is one of the types of inheritance.
- Syntax:

```
        class BaseClass1
        {
                        //body of BaseClass1 Class
        }
        class DerivedClass1 extends BaseClass1
        {
                        //body of DerivedClass1
        }
    class DerivedClass2 extends DerivedClass1
        {
                        //body of DerivedClass2
        }
```

-Example

```
//Multi-level Inheritance
import java.util.*;
class Student
{
        int rollno;
        String name;
        void get_stud_info()
        {
                Scanner sc=new Scanner(System.in);
                System.out.println("Enter Student Roll No:");
                rollno=sc.nextInt();
                System.out.println("Enter Student Name:");
                name=sc.next();
        }
        void disp_stud_info()
        {
```

```
                System.out.println("Student ROll No:"+rollno);
                System.out.println("Student Name:"+name);
        }
}
class Test extends Student
{
        int marks1,marks2;
        void get_stud_marks()
        {
                Scanner sc=new Scanner(System.in);
                System.out.println("Enter Student Test-1 Marks:");
                marks1=sc.nextInt();
                System.out.println("Enter Student Test-2 Marks:");
                marks2=sc.nextInt();
        }
        void disp_stud_marks()
        {
                System.out.println("Test-1 Marks:"+marks1);
                System.out.println("Test-2 Marks:"+marks2);
        }
}
class Result extends Test
{
        int total_marks;
        void get_total_marks()
        {
                total_marks=marks1+marks2;
        }
        void disp_total_marks()
        {
                System.out.println("Total Marks:"+total_marks);
        }
}

class MultilevelInheritanceDemo
{
        public static void main(String args[])
        {
                Result t1=new Result();
                t1.get_stud_info();
                t1.get_stud_marks();
                t1.get_total_marks();

                System.out.println("*******STUDENT INFORMATION SYSTEM********");
                t1.disp_stud_info();
                t1.disp_stud_marks();
                t1.disp_total_marks();
        }
}
/*
```

```
Enter Student Roll No:
1010
Enter Student Name:
James
Enter Student Test-1 Marks:
78
Enter Student Test-2 Marks:
90
*******STUDENT INFORMATION SYSTEM********
Student ROll No:1010
Student Name:James
Test-1 Marks:78
Test-2 Marks:90
Total Marks:168
*/
```

3) Multiple Inheritance:

- To create new class from more than one base class is known as multiple inheritance.
- But in java, we cannot derive multiple base classes properties in derived class.
- If you want to achieve this scenario then you can use alternate solution that is interface.
- Exmaple:

```
class A extends B extends C
{
        //body of A.
}
```

- Above scenario is not allowed in java language.

4) Hierarchical Inheritance:

- To create more than one derived classes from only one base class is known as Hierarchical inheritance.
- It is one of the types of inheritance.
- We use following syntax for creating the new class from old class.

```
class DerivedClassName extends BaseClassName
{
        //body of Derived Class
}
```

- Program:

```
//Hierarchical Inheritance
import java.util.*;
class Student
{
        int rollno;
        String name;
        void get_stud_info()
        {
```

```
                Scanner sc=new Scanner(System.in);
                System.out.println("Enter Student Roll No:");
                rollno=sc.nextInt();
                System.out.println("Enter Student Name:");
                name=sc.next();
        }
        void disp_stud_info()
        {
                System.out.println("Student ROll No:"+rollno);
                System.out.println("Student Name:"+name);
        }
}
class Test extends Student
{
        int marks1,marks2;
        void get_stud_marks()
        {
                Scanner sc=new Scanner(System.in);
                System.out.println("Enter Student Test-1 Marks:");
                marks1=sc.nextInt();
                System.out.println("Enter Student Test-2 Marks:");
                marks2=sc.nextInt();
        }
        void disp_stud_marks()
        {
                System.out.println("Test-1 Marks:"+marks1);
                System.out.println("Test-2 Marks:"+marks2);
        }
}
class Sport extends Student
{
        float sport_wt;
        void get_sport_info()
        {
                sport_wt=8.9f;
        }
        void disp_sport_info()
        {
                System.out.println("Sport Weightage:"+sport_wt);
        }
}

class HierarchicalInheritanceDemo
{
        public static void main(String args[])
        {
                Test t1=new Test();
                System.out.println("*******Test Class Implementation********");
                t1.get_stud_info();
                t1.get_stud_marks();
```

```
                t1.disp_stud_info();
                t1.disp_stud_marks();

                Sport s1=new Sport();
                System.out.println("*******Sport Class Implementation********");
                s1.get_stud_info();
                s1.get_sport_info();
                s1.disp_stud_info();
                s1.disp_sport_info();

        }
}
/*
*******Test Class Implementation********
Enter Student Roll No:
1010
Enter Student Name:
James
Enter Student Test-1 Marks:
89
Enter Student Test-2 Marks:
78
Student ROll No:1010
Student Name:James
Test-1 Marks:89
Test-2 Marks:78
*******Sport Class Implementation********
Enter Student Roll No:
1010
Enter Student Name:
James
Student ROll No:1010
Student Name:James
Sport Weightage:8.9
*/
```

5) Hybrid Inheritance:

- The combination of more than one types inheritance is known as hybrid inheritance.
- Example:

```
//Hybrid Inheritance
import java.util.*;
class Student
{
        int rollno;
        String name;
        void get_stud_info()
        {
                Scanner sc=new Scanner(System.in);
```

```
                System.out.println("Enter Student Roll No:");
                rollno=sc.nextInt();
                System.out.println("Enter Student Name:");
                name=sc.next();
        }
        void disp_stud_info()
        {
                System.out.println("Student ROll No:"+rollno);
                System.out.println("Student Name:"+name);
        }
}
class Test extends Student
{
        int marks1,marks2;
        void get_stud_marks()
        {
                Scanner sc=new Scanner(System.in);
                System.out.println("Enter Student Test-1 Marks:");
                marks1=sc.nextInt();
                System.out.println("Enter Student Test-2 Marks:");
                marks2=sc.nextInt();
        }
        void disp_stud_marks()
        {
                System.out.println("Test-1 Marks:"+marks1);
                System.out.println("Test-2 Marks:"+marks2);
        }
}
class Result extends Test
{
        int total_marks;
        void get_total_marks()
        {
                total_marks=marks1+marks2;
        }
        void disp_total_marks()
        {
                System.out.println("Total Marks:"+total_marks);
        }
}
class Sport extends Student
{
        float sport_wt;
        void get_sport_info()
        {
                sport_wt=8.9f;
        }
        void disp_sport_info()
        {
                System.out.println("Sport Weightage:"+sport_wt);
```

```
        }
}
class HybridInheritanceDemo
{
        public static void main(String args[])
        {
                System.out.println("*******RESULT CLASS IMPLEMENTATION********");
                Result r1=new Result();
                r1.get_stud_info();
                r1.get_stud_marks();
                r1.get_total_marks();

                r1.disp_stud_info();
                r1.disp_stud_marks();
                r1.disp_total_marks();

                System.out.println("*******SPORT CLASS IMPLEMENTATION********");
                Sport s1=new Sport();
                s1.get_stud_info();
                s1.get_sport_info();
                s1.disp_stud_info();
                s1.disp_sport_info();
        }
}
/*
*******RESULT CLASS IMPLEMENTATION********
Enter Student Roll No:
1010
Enter Student Name:
James
Enter Student Test-1 Marks:
78
Enter Student Test-2 Marks:
99
Student ROll No:1010
Student Name:James
Test-1 Marks:78
Test-2 Marks:99
Total Marks:177
*******SPORT CLASS IMPLEMENTATION********
Enter Student Roll No:
1010
Enter Student Name:
James
Student ROll No:1010
Student Name:James
Sport Weightage:8.9
*/
```

Method Overriding:

==========
- Suppose, base class and derived class method names are same.
- When base class method derived in derived class then it got overide.
- It means base class method overriden by derived class method.
- To call overriden method, we can use super keyword.
- We use syntax for calling hidden method: super.methodName();

- Program:

```
//Method overriding and use of super keyword
class Base
{
        void display()
        {
                System.out.println("display() method of base class");
        }
}
class Derived extends Base
{
        void display()
        {
                super.display();
                System.out.println("display() method of derived class");
        }
}
class MethodOverriding
{
        public static void main(String args[])
        {
                Derived d1=new Derived();
                d1.display();
        }
}
/*
display() method of base class
display() method of derived class
*/
```

==========
How to invoke Base class Constructor
==========
- Base class constructor should not inherited in its sub class.
- Suppose, base class contain constructor then how we can call that constructor.
- In this case , we can super keyword.
- super keyword should be the first line of derived class constructor body.
- Syntax:

```
                super();                                //to invoke default
constructor
                or
                super(argument list)    //to invoke parameterized constructor
```

-Program1:

```
//use of super keyword for calling base class default constructor.
class Base
{
        Base()
        {
                System.out.println("Base class constructor called...!!!");
        }
}
class Derived extends Base
{
        Derived()
        {
                super();
                System.out.println("Derived class constructor called...!!!");

        }
}
class InvokeBaseClassConstructor
{
        public static void main(String args[])
        {
                Derived d1=new Derived();
        }
}
/*
Base class constructor called...!!!
Derived class constructor called...!!!
*/
-Program2:
//use of super keyword for calling base class parameterized constructor.
class Base
{
        int x;
        Base(int m)
        {
                x=m;
                System.out.println("Base class constructor called..m="+m);
        }
}
class Derived extends Base
{
        int y;
        Derived(int p,int q)
        {
                super(p);
                y=q;
                System.out.println("Derived class constructor called..q="+q);

        }
}
```

```
class InvokeBaseClassConstructor1
{
        public static void main(String args[])
        {
                Derived d1=new Derived(100,200);
        }
}
/*
Base class constructor called..x=100
Derived class constructor called..y=200
*/
```

```
=======================================
***State three uses of final keyword***
=======================================
- final is a predefined keyword.
- Following are the uses of final keyword:
1) To make constant variable:
-----------------------------
- If we declare the variable using final keyword then that variable become constant
in java.
- Constant variable means, once it is created then we can not change its value
later.
- Constant us a variable which can not change its value during the execution of
program.
Example:
class finalKeywordDemo
{
        public static void main(String args[])
        {
                final float PI=3.14f;           //constant variable
                int radius=2;
                float area;
                area=(PI*radius*radius);
                System.out.println("Area of Circle="+area);
        }
}
/*
Area of Circle=12.56
*/

2) To avoid method overriding:
------------------------------
- Suppose, base class method name and derived class method name are same then base
class method overriden by derived class method.
- If we want to avoid method overriding then we can use final keyword before the
base class method declaration.
- Example:
//To avoid method overriding
class Base
{
        final void display()
        {
                System.out.println("display method of base class");
        }
}
class Derived extends Base
{
        void display()
        {
                System.out.println("display method of derived class");
        }
```

```
}
class AvoidMethodOverriding
{
        public static void main(String args[])
        {
                Derived d1=new Derived();
                d1.display();
        }
}
/*
AvoidMethodOverriding.java:11: error: display() in Derived cannot override
display() in Base
        void display()
             ^
  overridden method is final
1 error
*/
```

3) To avoid inheritance:

- To avoid inheritance then we can declare the base class using final keyword.
- If we declare base class using final keyword then we can not create subclass from it.
- Example:

```
//To avoid method overriding
final class Base
{
        void display()
        {
                System.out.println("display method of base class");
        }
}
class Derived extends Base
{
        void show()
        {
                System.out.println("show method of derived class");
        }
}
class AvoidInheritance
{
        public static void main(String args[])
        {
                Derived d1=new Derived();
                d1.display();
                d1.show();
        }
}
/*
AvoidInheritance.java:9: error: cannot inherit from final Base
```

```
class Derived extends Base
                      ^
1 error
*/
```

❖ Interface:

- An interface is similar to the class.
- Interface is a collection of abstract methods and final static variables.
- Interface is used to achieve the multiple inheritance concepts in Java.
- All the abstract methods of the interface need to be defined in its sub class.
- An Interface does not contain any constructors.
- We cannot create the objects of an interface.
- All of the methods of the Interface are by default abstract.
- All of the variables of the Interface are by default static, final.
- An interface is not extends the properties of the class but it is implemented by a class.
- One interface can extend the multiple interfaces.

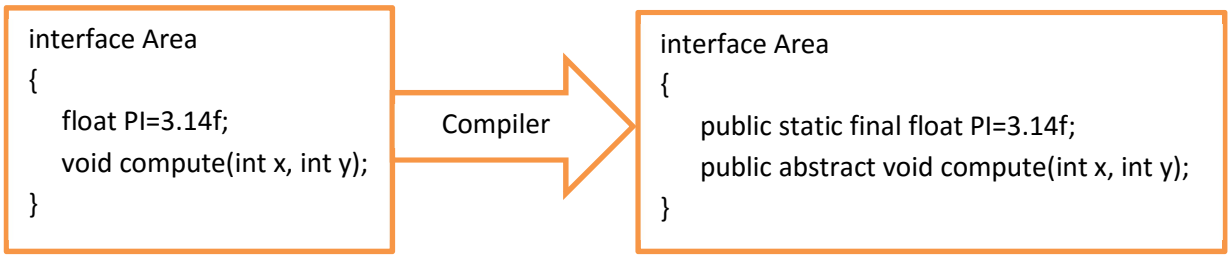
Syntax:

```
interface Interface_Name
{
        datatype final_variable_name1=value1; //Any number of final,static variables
        ---
        ---
        returntype method_name(parameter_list); //Any number of abstract method declarations
        ---
        ---
}
```

Example:

```
interface Area
{
        float PI=3.14f;
        void compute(int x,int y);
}
```

The java compiler adds public and abstract keywords before the interface method. It adds public, static and final keywords before data members.



Implementing interface:

- Class implements an interface.
- The class uses the **implements** keyword for implementing the interface.
- Class is responsible for implementing the abstract methods of the interface.

Syntax:

```
class class_name implements interface_name
{
    //body of class
}
```

Example:

```
interface Area
{
       float  PI=3.14f;
       void compute(int  r);
}

class Circle implements Area
{
      public void compute(int r)
      {
            System.out.println("Area of Circle="+(PI*r*r));
      }
      public static void main(String args[])
      {
           Circle c1=new Circle();
           c1.compute(10);
      }
}
```

Output:

```
Area of Circle=314.0
```

Extending interface:

- One interface can extend another interface same way that a class can extend another class.
- The **extends** keyword is used to extend an interface.
- In this case sub interface inherits the properties of super interface, but it will not define the methods of the super interface.
- Java class does not extend more than one class because java does not support to the multiple inheritance.
- But interface is not a class, it can extend two or more interfaces, they are separated by the commas.

Syntax:

```
interface sub_interface extends super_interface
{
       //body of sub interface
}
```

Example:

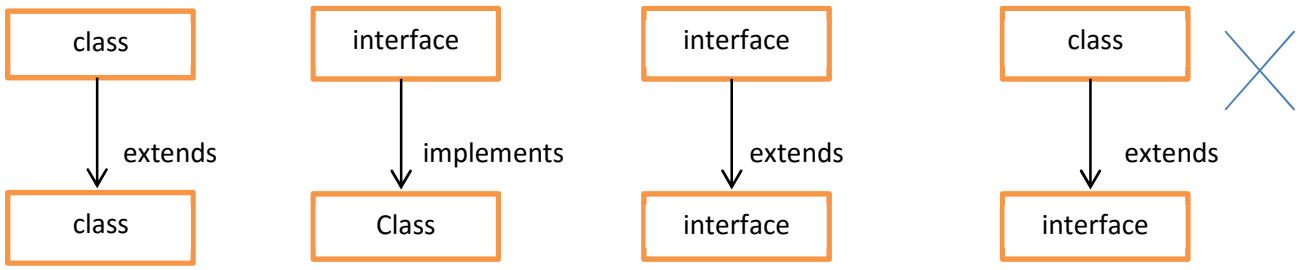
```
interface Abc
{
       void display();
}

interface Xyz extends Abc
{
      void show();

}
class Mnp implements Xyz
{
       public void display()
       {
                System.out.println("I am from Abc interface:");
       }
       public void show()
       {
                System.out.println("I am from Xyz interface:");
       }
       public static void main(String args[])
       {
                Mnp m1=new Mnp();
                m1.display();
                m1.show();
       }
}
```

Output:

```
I am from Abc interface:
I am from Xyz interface:
```

❖ Understand the relationship between classes and interfaces:

class extends class class implements interface interface extends interface interface extends class



❖ Multiple inheritance in Java by using interface:

- To inherit the properties of more than one base class into sub class is known as multiple inheritances.
- In multiple inheritance we can combine the features of more than one existing classes into new class.
- Below diagram shows the multiple inheritance concepts:

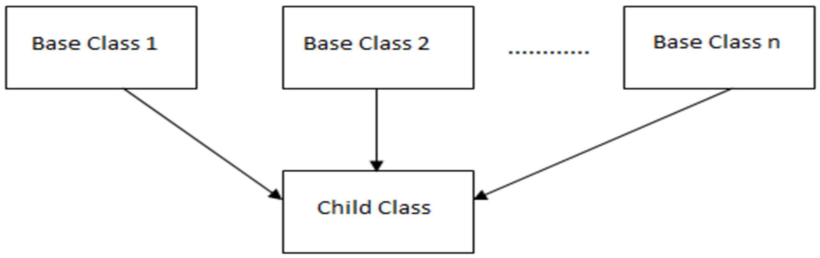


Fig: Multiple Inheritance

- Java classes cannot have more than one super class. But in most of the real time application multiple inheritances is required. So java provides an alternative approach is known as interface.
- Interface is a collection of static final variables and abstract methods. It is used to achieve the multiple inheritance in Java.
- If a class implements multiple interfaces, or an interface extends multiple interfaces i.e. known as multiple inheritance in Java.
- Below diagram shows, How to achieve the multiple inheritance in Java language:

Multiple Inheritance in Java

- In first diagram, class implementing more than one interface and in second diagram interface extending more than one interfaces to achieve the multiple inheritance in Java.

Example:

```
interface Abc
{
       void display();
}

interface Xyz
{
      void show();

}
class Mnp implements Abc,Xyz
{
       public void display()
        {
                 System.out.println("I am from Abc interface:");
        }
       public void show()
        {
                 System.out.println("I am from Xyz interface:");
        }
       public static void main(String args[])
       {
                 Mnp m1=new Mnp();
                 m1.display();
                 m1.show();
       }
}
```

Output:

```
I am from Abc interface:
I am from Xyz interface:
```

❖ Difference between Abstract class and Interface:

Abstract class	Interface
1) Abstract class can have abstract and non-abstract methods.	Interface can have only abstract methods.
2) Abstract class doesn't support multiple inheritance.	Interface supports multiple inheritance.
3) Abstract class can have final, non-final, static and non-static variables.	Interface has only static and final variables.
4) Abstract class can provide the implementation of interface.	Interface can't provide the implementation of abstract class.
5) The abstract keyword is used to declare abstract class.	The interface keyword is used to declare interface.
6) Abstract class can extend another Java class and implements multiple interfaces.	An interface can extend another Java interface only.
7) Abstract class achieves partial abstraction.	Interface achieves fully abstraction.

8) Example:
abstract class Abc
{
abstract void draw();
}
	Example:
	interface Xyz
	{
	void draw();
	}

❖ Difference between class and Interface:

Class	Interface
1) We can create objects from class.	We cannot create the objects from interface.
2) The members of a class can be private, public or protected.	The members of an interface are always public.
3) Class contains abstract or non-abstract methods.	Interface has only abstract methods.
4) Class can contain methods definitions.	Interface can contain only declaration of methods without body.
5) The class keyword is used to declare class.	The interface keyword is used to declare interface.
6) Class can implement any number of interfaces and can extend only one class.	An interface can extend multiple interfaces but cannot implement any class.
7) Constructor present in class	Constructor not present in class

8) Example:
class Abc
{
void draw();
}
	Example:
	interface Xyz
	{
	void draw();
	}

❖ Java Package:

- Putting classes and interfaces together is known as Package.
- Package is a collection of similar types of classes and interfaces.
- Packages are acts as container for the classes and interfaces.
- We can achieve the reusability features of Java by using Inheritance and packages.
- Package in Java is a mechanism to encapsulate a group of classes, sub packages and interfaces.
- A package is a collection of related classes. It helps Organize your classes into a folder structure and make it easy to locate and use them. More importantly, it helps improve re-usability.
- Each package in Java has its unique name and organizes its classes and interfaces into a separate namespace, or name group.

Advantage of Java Package:

- Java package is used to categorize the classes and interfaces so that they can be easily maintained.
- Java package provides access protection.
- Java package removes naming collision.
- The classes present in the packages of other programs can be reused.
- Package provides the good ways of hide the classes.
- In package, classes can be unique i.e. two classes in two different packages can have the same name.

There are two types of packages in Java:

1. Java API packages/Built-in packages.
2. User Defined packages.

❖ Java API Packages/Built-in Packages:

- Java API provides a number of classes grouped into different packages according to their functionality.
- The already defined package like java.io.*, java.lang.* etc. are known as built-in packages.
- Below diagram shows Java API packages:

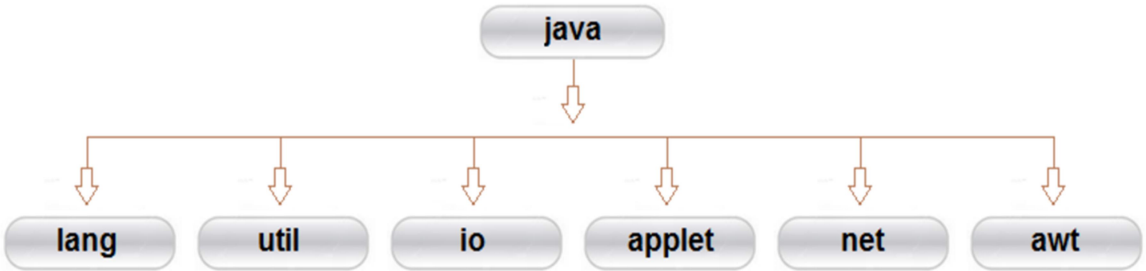


Fig. Java API Packages

- There are six main packages are present in Java programming language.
- Following table gives the information about the Java system packages and their classes

Package Name	Contents of the package
java.lang	Language support classes. They include classes for primitive types, string, math functions, thread and exceptions.
java.util	Language utility classes such as vectors, hash tables, random numbers, data, etc.
java.io	Input/output support classes. They provide facilities for the input and output of data.
java.applet	Classes for creating and implementing applets.
java.net	Classes for networking. They include classes for communicating with local computers as well as with internet servers.
java.awt	Set of classes for implementing graphical user interface. They include classes for windows, buttons, lists, menus and so on.

❖ User defined Packages:

- Creating a package in java is quite easy.
- A package is always defined in a separate folder having the same name as a package name.
- A package is a collection of related classes. It helps Organize your classes into a folder structure and make it easy to locate and use them. More importantly, it helps improve re-usability.
- Each package in Java has its unique name and organizes its classes and interfaces into a separate namespace, or name group.
- The package must contain one or more classes or interfaces. This implies that a package cannot be empty.
- The classes or interfaces that are in the package must have their source files in the same directory structure as the name of the package.
- A package is always defined in a separate folder having the same name as a package name.
- Define all classes in that package folder.
- All classes of the package which we wish to access outside the package must be declared public.
- All classes within the package must have the package statement as its first line.
- All classes of the package must be compiled before use (So that its error free)

Creation of packages includes following steps:

1. Declare the package at the beginning of the java source code file using below syntax.

Syntax:

```
package package_name;
```

Above statement should be used in the beginning of the program to include that program in that particular package.

2. Define a class which is to be put in the package and declare it public.

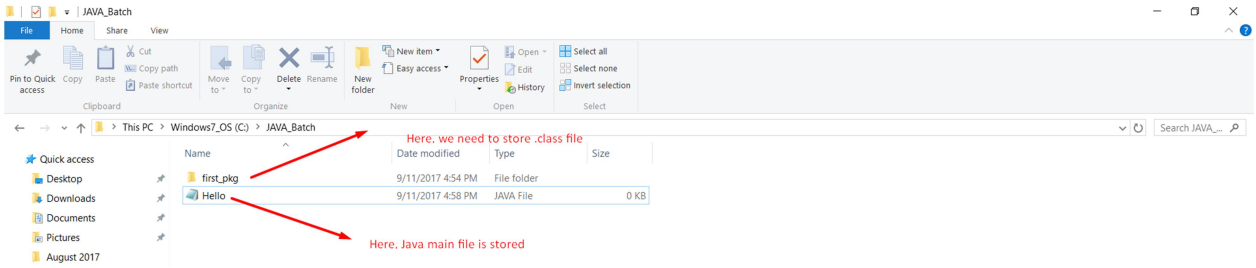
Example:

```
package first_pkg;

public class Hello
{
        public void display()
        {
                System.out.println("I am from Hello class:");
        }
}
```

In above example **first_pkg** is the package name and class Hello is added into it.

3. Create the subdirectory which has given the same name of package and stored it into main directory where the source code is present.

4. Compile the source file, after that .class file stored into the subdirectory which is created in step 3.



❖ How to access package from another package/import statement:

There are three ways to access the package from outside the package.

1. import package.*;
2. import package.classname;
3. fully qualified name.

1) Using packagename.*;

- If you use package.* statement then all the classes and interfaces of this package will be accessible.
- import is a predefined keyword.

Syntax:

```
import Package_Name.*;
```

Example:

```
//Save by Abc.java

package first_pkg;

public class Abc
{
	public void display()
	{
		System.out.println("I am from Abc class:");
	}
}
```

```
//Save by Xyz.java

package second_pkg;
import first_pkg.*;

class Xyz
{
	public static void main(String args[])
	{
		Abc a1=new Abc();
		a1.display();
	}
}
```

Output:

```
I am from Abc class
```

2) import packagename.classname;

- If you use package.classname statement then only declared class of that package will be accessible.
- import is a predefined keyword.

Syntax:

```
import Package_Name.classname;
```

Example:

```
//Save by Abc.java

package first_pkg;

public class Abc
{
	public void display()
	{
		System.out.println("I am from Abc class:");
	}
}
```

```
//Save by Xyz.java

package second_pkg;
import first_pkg.Abc;

class Xyz
{
	public static void main(String args[])
	{
		Abc a1=new Abc();
		a1.display();
	}
}
```

Output:

```
I am from Abc class
```

3) Using fully qualified name;

- If you use fully qualified name then only declared class of this package will be accessible.
- No need to use the import keyword.
- But you need to use fully qualified name every time when you are accessing the class or interface.
- It is generally used when two packages have same class name e.g. java.util and java.sql packages contain Date class.

Example:

```
//Save by Abc.java

package first_pkg;

public class Abc
{
	public void display()
	{
		System.out.println("I am from Abc class:");
	}
}
```

```
//Save by Xyz.java

package second_pkg;

class Xyz
{
	public static void main(String args[])
	{
		first_pkg.Abc a1=new first_pkg.Abc ();  //using fully qualified name
		a1.display();
	}
}
```

Output:

```
I am from Abc class
```

❖ Java Static Import

- The static import feature of Java is used to access any static member of a class directly.
- There is no need to use the class name while accessing the static members.
- The static import statement can be used to import static members from classes and use them without using class name.
- The difference between import and static import: - The import allows the java programmer to access classes of a package without using package name whereas the static import feature allows accessing the static members of a class without using the class name.
- The import provides accessibility to classes and interface whereas static import provides accessibility to static members of the class.

Syntax:

```
import static Package_Name.*;
          or
import static Package_Name.classname;
```

Example:

```
import static java.lang.System.*;

class StaticImportDemo
{
         public static void main(String args[])
         {

                  out.println("Hello"); //Now no need of System.out
                  out.println("Java");
         }
}
```

Output:

```
Hello
Java
```

Advantage:

- Less coding is required if you have access any static member of a class.

Disadvantage:

- If you overuse the static import feature, it makes the program unreadable and unmaintainable.

Package:
========
- Creating a package in java is quite easy.
- A package is always defined in a separate folder having the same name as a package name.
- A package is a collection of related classes. It helps Organize your classes into a folder structure and make it easy to locate and use them. More importantly, it helps improve re-usability.
- Each package in Java has its unique name and organizes its classes and interfaces into a separate namespace, or name group.
- The package must contain one or more classes or interfaces. This implies that a package cannot be empty.
- The classes or interfaces that are in the package must have their source files in the same directory structure as the name of the package.
- A package is always defined in a separate folder having the same name as a package name.
- Define all classes in that package folder.
- All classes of the package which we wish to access outside the package must be declared public.
- All classes within the package must have the package statement as its first line.
- All classes of the package must be compiled before use (So that its error free)

Steps for creation of package:
----
1) Write below line as first line of java source file
        package packageName;
2) Write java class which you want to add inside the package and make it as public
3) Create directory whose name same as package name and stored java file inside it and compile it.
4) Access created package in another program using below different ways
        import packageName.*;
        import packageName.className;
        fully qualified name

Example:

```
//Declare package msbte and make the class public
package msbte;
public class Sample
{
        public void display()
        {
                System.out.println("display method of Sample class");
        }
}

//Accessing msbte package
import msbte.*;
class AccessSamplePKG
{
```

```
        public static void main(String args[])
        {
                Sample s1=new Sample();
                s1.display();
        }
}
```

OUTPUT:
========
display method of Sample class

==========
Write a program to create package Math_s having two classes as addition and subtraction. Use suitable methods in each class to perform basic operations. *
==========

```
//Creating Math_s package
package Math_s;
public class Addition
{
        public void add(int x,int y)
        {
                System.out.println("Addition="+(x+y));
        }
}
public class Subtraction
{
        public void sub(int x,int y)
        {
                System.out.println("Subtraction="+(x-y));
        }
}
//Accessing Math_s package
import Math_s.*;
class AccessMathPKG
{
        public static void main(String args[])
        {
                Addition a1=new Addition();
                Subtraction s1=new Subtraction();
                a1.add(100,50);
                s1.sub(500,300);
        }
}
```

❖ Multithreading in Java:

- The process of executing multiple threads simultaneously is known as Multithreading.
- Multithreading is a Java feature that allows concurrent execution of two or more parts of a program. Each part of such program is called a thread
- Thread is basically a lightweight process.
- Multiple threads share a common memory area for their execution.
- They don't allocate separate memory space for each thread. That's via it is called as light weight process.
- Process is a heavy weight so we use Multithreading than Multiprocessing.
- Context-switching between the threads takes less time than process.
- Java Multithreading is mostly used in games, animation etc.
- Due to Multi-threading, multiple activities can proceed concurrently in the same program.
- Multitasking is a process of executing multiple tasks simultaneously. We use multitasking to utilize the CPU.
- Multitasking can be achieved by two ways:
  1. Process-based Multitasking (Multiprocessing)
  2. Thread-based Multitasking (Multithreading)
- Below diagram shows the Java Multithread program

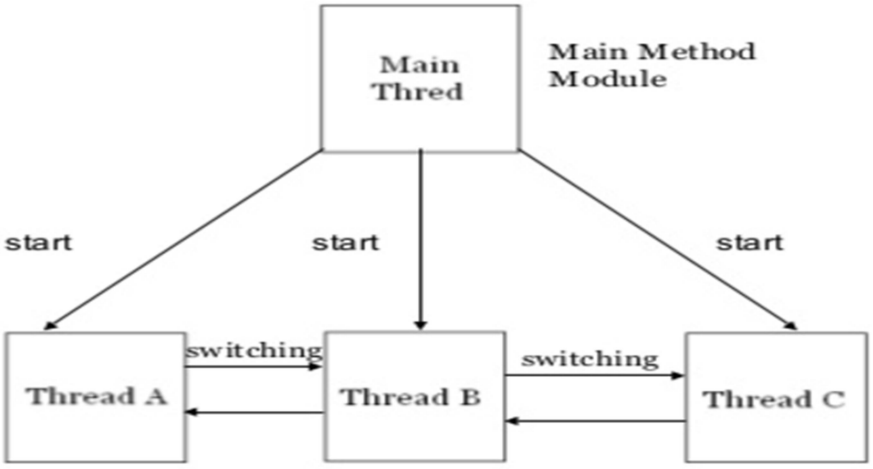


Fig. Java Multithread Program

Advantages of Java Multithreading:

- It doesn't block the user because threads are independent and you can perform multiple operations at same time.
- You can perform many operations together so it saves time.
- Threads are independent so it doesn't affect other threads if exception occurs in a single thread.

❖ Difference between Multithreading and Multiprocessing:

Multithreading	Multiprocessing
1) The process of executing multiple threads simultaneously is known as Multithreading	The process of executing multiple process simultaneously is known as Multiprocessing
2) Thread is lightweight	Process is heavyweight
3) Threads share the same address space	Each process allocates separate memory space
4) Cost of communication between the thread is low	Cost of communication between the process is high
5) Same job is divided into multiple threads and executed simultaneously.	Multiple jobs can execute simultaneously.
6) Creation of thread is not time-consuming	Creation of a process is time-consuming
7) Multithreading is not classified.	Multiprocessing can be symmetric or asymmetric.
8) It saves processor time and memory.	It does not save processor time and memory.

❖ Life cycle of a Thread:

- A thread goes through various stages in its life cycle. The life cycle of the thread in java is controlled by JVM.
- The following diagram shows the complete life cycle of a thread.

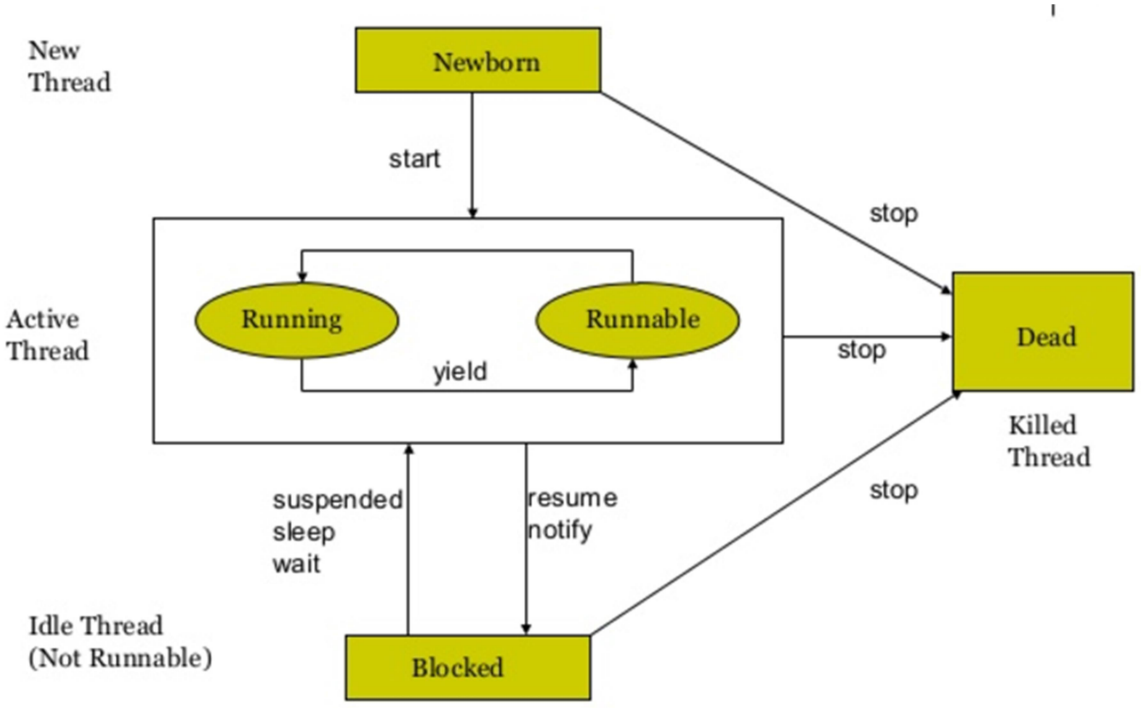


Fig. Life Cycle of Thread.

Following are the stages of the life cycle of a Thread.

1. Newborn state
2. Runnable state
3. Running state
4. Blocked state
5. Dead state

1. Newborn state:

- When a thread object is created, the thread is born then called as thread is in Newborn state.
- A new thread begins its life cycle in this state.
- When thread is in Newborn state then you can pass to the Running state by invoking start() method or kill it by using stop() method.

2. Runnable state:

- The thread is in runnable state after invocation of start() method.
- The Runnable state of thread means that the thread is ready for the execution and waiting for the availability of the processor.
- The threads which are ready for the execution are managed in the queue.
- The same priority threads are processed on the basis of First-come-First-Serve.

3. Running state:

- The Running state means that the processor has given it's time to thread for their execution
- When thread is executing, its state is changed to Running.
- A thread can change state to Runnable, Dead or Blocked from running state depends on time slicing, thread completion of run() method or waiting for some resources.

4. Blocked state:

- A thread can be temporarily suspended or blocked from entering into runnable or running state.
- Thread can be blocked due to waiting for some resources to available.
- Thread can be blocked by using following thread methods:
  I. suspended()
  II. wait()
  III. sleep()
- Following are the methods used to entering thread into Runnable state from Blocked state.
  I. The resume() method is invoked in case of suspended().
  II. The notify() method is invoked in case of wait().
  III. When the specified time is elapsed in the case of sleep().

5. Dead state:

- The thread will move to the dead state after completion of its execution. It means thread is in terminated or dead state when its run() method exits.
- Also Thread can be explicitly moved to the dead state by invoking stop() method.

❖ Creating a Thread in Java:

There are two different ways of creating the Thread in Java programming language.

1. By extending Thread class.
2. By implementing Runnable interface.

1. By extending the Thread class:

We can create the thread by extending the Thread class. Thread class provide constructors and methods to create and perform operations on a thread.

Constructors of Thread class:

- Thread()
- Thread(String name)
- Thread(Runnable r)
- Thread(Runnable r, String name)

Following steps are required to create the thread in Java by using Thread class:

I. Declare the class by extending the Thread class.

```
class ThreadX extends Thread
{
    //body of ThreadX class
}
```

II. Implement the run() method.

```
public void run()
{
    //thread code
}
```

III. Create the object of a thread class.

```
ThreadX t1=new ThreadX();
```

IV. Invoke the start() method.

```
t1.start();
```

Example:

```
//*********************************************************************************************
class ThreadX extends Thread
{
    public void run()
    {
          for(int i=1;i<=5;i++)
          {
                    System.out.println("From ThreadX:i="+i);
          }
    }
}

//*********************************************************************************************
class ThreadY extends Thread
{
    public void run()
    {
          for(int j=1;j<=5;j++)
          {
                    System.out.println("From ThreadY:j="+j);
          }
    }
}

//*********************************************************************************************
class ThreadDemo
{
    public static void main(String args[])
    {
          ThreadX t1=new ThreadX();
          ThreadY t2=new ThreadY();
          t1.start();
          t2.start();
    }
}
```


2. By implementing Runnable interface.

We can create the thread by implementing the Runnable interface. Following steps are required to create the Thread in Java.

I. Declare the class by implementing the Runnable interface.

```
class RunnableX implements Runnable
{
    //body of ThreadX class
}
```

II. Implement the run() method.

```
public void run()
{
    //thread code
}
```

III. Create the object of a Thread class by passing object of class as argument which is implemented from the Runnable interface.

```
RunnableX r1=new RunnableX ();
Thread t1=new Thread(r1);
```

IV. Invoke the start() method.

```
t1.start();
```

Example:

```
//*************************************************************************************
class RunnableX implements Runnable
{
    public void run()
    {
          for(int i=1;i<=5;i++)
          {
                    System.out.println("From RunnableX:i="+i);
          }
    }
}

//*************************************************************************************
class RunnableY implements Runnable
{
    public void run()
    {
          for(int j=1;j<=5;j++)
          {
                    System.out.println("From RunnableY:j="+j);
          }
    }
}

//*************************************************************************************
class RunnableDemo
{
    public static void main(String args[])
    {
          RunnableX r1=new RunnableX();
          RunnableY r2=new RunnableY();

          Thread t1=new Thread(r1);
          Thread t2=new Thread(r2);
```

```
        t1.start();
        t2.start();
    }
}
```

❖ **Important points between Thread Class vs. Runnable Interface**

1. If we extend the Thread class, our class cannot extend any other class because Java doesn't support multiple inheritance. But, if we implement the Runnable interface, our class can extend other base classes.
2. We can achieve basic functionality of a thread by extending Thread class because it provides some inbuilt methods like yield(), interrupt() etc. that are not available in Runnable interface.

❖ **Thread priority:**

- Each thread have a priority, Priorities are represented by a number between 1 and 10.
- Thread priorities are the integers which decide how one thread should be treated with respect to the others.
- Thread priority decides when to switch from one running thread to another, process is called context switching
- A thread can voluntarily release control and the highest priority thread that is ready to run is given the CPU.
- A thread can be preempted by a higher priority thread no matter what the lower priority thread is doing. Whenever a higher priority thread wants to run it does.
- To set the priority of the thread below setPriority() method is used which is a method of the class Thread.

```
ThreadName.setPriority(int number);
```

Where, number is an integer value which between 1 to 10.

- getPriority() is used to retrieve the priority of the thread.
- In place of defining the priority in integers, we can use below three constants:
  - **MIN_PRIORITY**
  - **NORM_PRIORITY**
  - **MAX_PRIORITY**
- Default priority of a thread is 5 (NORM_PRIORITY). The value of MIN_PRIORITY is 1 and the value of MAX_PRIORITY is 10.

Example:

```
//*********************************************************************************************
class ThreadX extends Thread
{
    public void run()
    {
         for(int i=1;i<=5;i++)
         {
                  System.out.println("From ThreadX:i="+i);
         }
    }
}

//*********************************************************************************************
class ThreadY extends Thread
{
    public void run()
    {
         for(int j=1;j<=5;j++)
         {
                  System.out.println("From ThreadY:j="+j);
         }
    }
}

//*********************************************************************************************
class ThreadZ extends Thread
{
    public void run()
    {
         for(int K=1;K<=5;K++)
         {
                  System.out.println("From ThreadZ:K="+K);
         }
    }
}


//*********************************************************************************************
class ThreadDemo
{
    public static void main(String args[])
    {
         ThreadX t1=new ThreadX();
         ThreadY t2=new ThreadY();
         ThreadZ t2=new ThreadZ();

         t1.setPriority(Thread.MIN_PRIORITY);
         t2.setPriority(Thread.NORM_PRIORITY);
         t3.setPriority(Thread.MAX_PRIORITY);

         t1.start();
         t2.start();
         t3.start();
    }
}
```

❖ Exception handling in Java:

- Exception handling is one of the most important features of Java programming.
- It handles the runtime errors caused by exceptions.

Exception:

- An Exception is an unwanted event that interrupts the normal flow of the program.
- Exception may or may not be occur in program.
- When an exception occurs program execution gets terminated. In this case we get a system generated error message.
- When Java interpreter found the error such as divide by zero, the interpreter creates an exception object and throws it to inform that exception is occurred.
- The good thing about the exceptions is that they can be handled in Java.
- By handling the exceptions we can provide a meaningful message to the user about the issue rather than a system generated message.
- There can be several reasons that can cause a program to throw exception.
- For example: Opening a non-existing file in your program, Network connection problem, bad input data provided by user, divide by zero etc.

Exception Handling:

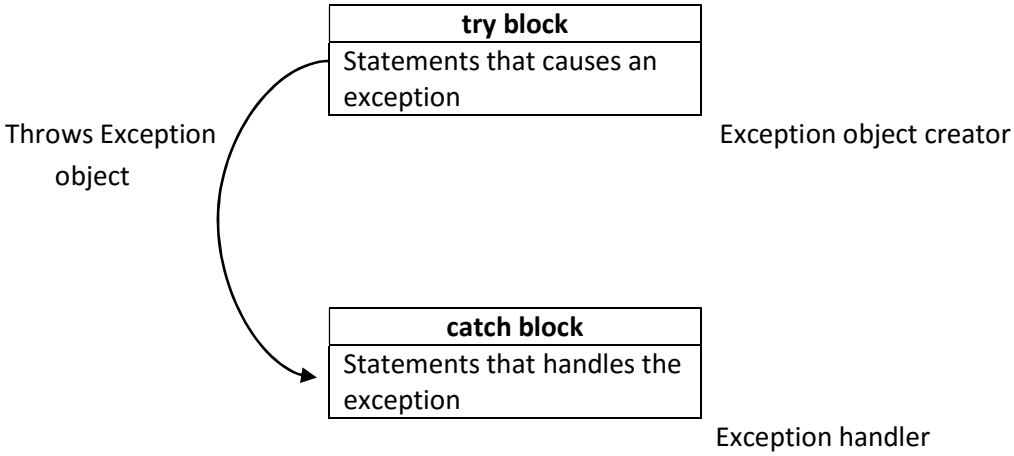
- Exception Handling is a mechanism to handle the runtime errors.
- If an exception occurs, which has not been handled by programmer then program execution gets terminated and a system generated error message is shown to the user.
- For example look at the system generated exception below:

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at ExceptionDemo.main(ExceptionDemo.java:7)
```

- This message is not user friendly so a user will not be able to understand.
- So by using Exception handling mechanism, we handle an Exception and then print a user friendly warning message to user.
- The basic idea of exception handling mechanism is to find the exception, Throws the exception, Catch the exception and handle the exception.
- There are 5 keywords used in java exception handling.

1. try
2. catch
3. finally
4. throw
5. throws

- Following diagram shows the exception handling mechanism:



- **Advantages of Exception Handling:**
  1. The main advantage of exception handling is to maintain the normal flow of the application.
  2. Exception normally disrupts the normal flow of the application that via we use exception handling. Let's take a scenario:

```
Statement 1;
Statement 2;
Statement 3;
Statement 4; //Exception occurs
Statement 5;
Statement 6;
```

Suppose there is 6 statements in your program and there occurs an exception at statement 4, rest of the code will not be executed i.e. statement 5 to 6 will not run. If we perform exception handling, rest of the statement will be executed. That is why we use exception handling in java.

- **There are 5 keywords used in java exception handling.**

1. **try** – The statements which cause the exception are given in try block. The try block contains set of statements where an exception can occur. If exception is occurred then exception object is created and it is thrown. Syntax of try block is given below:

```
try
{
        //statements that may cause an exception
}
```

While writing a program, if you think that certain statements in a program can throw an exception, enclosed them in try block and handles that exception

2. **catch** – The statements which handles the exception are given in catch block. This block catches the exception which is generated by the try block and handles it. This block must write after the try block. A single try block can have several catch blocks. When an exception occurs in try block, the corresponding catch block that handles that particular exception executes. Syntax of catch block:

```
try
{
	//statements that may cause an exception
}
catch
{
	// Statements which handles the exception
}
```

3. **throw** – System generated exceptions are automatically thrown by the Java runtime system. To manually throw an exception 'throw' keyword is used. The throw keyword in Java is used to explicitly throw an exception from a method or any block of code.

```
throw ExceptionObject;
```

4. **throws** - throws is a keyword in Java which is used to indicate that this method might throw one of the listed type exceptions. Below is the general form of a method which includes a throws clause:

```
returntype method_name(parameters) throws exception_list
{
	//method body
}
```

5. **finally** – finally is a keyword in Java. finally block is always executed whether exception is handled or not. We can write the finally block after the try block or catch block. Finally block is optional.

```
try
{
	//statements that may cause an exception
}
catch
{
	// Statements which handles the exception
}
finally
{
	//Statements to be executed
}
```

- **Example:**

```
class ExceptionDemo
{
  public static void main(String args[])
  {
          int x,y,a=10,b=5,c=5;
          try
          {
                  System.out.println("try block execution begin");
                  x=a/(b-c);
                  System.out.println("try block execution end");
          }
          catch(ArithmeticException e)
          {
                  System.out.println("Divide by zero error is occurred");
          }
          finally
          {
                  System.out.println("Statements always executed");

          }
          y=a/(b+c);
          System.out.println("Y="+y);
  }
}
```

Output

```
try block execution begin
Divide by zero error is occurred
Statements always executed
Y=1
```

❖ **Multiple catch blocks:**

- A single try block can have several catch blocks. When an exception occurs in try block, then corresponding catch block which handles that particular exception is executed.
- Below syntax is used to write the multiple catch block:

```
try
{
  //statements that may cause an exception
}
catch
{
  // Statements which handles the exception
}
catch
{
```

```
  // catch block Statements 1
}
catch
{
  // catch block Statements 2

}
---
---
```

- **Example:**

```
class ExceptionDemo
{
  public static void main(String args[])
  {
          int x,y,a=10,b=5,c=5;
          try
          {
                  System.out.println("try block execution begin");
                  x=a/(b-c);
                  System.out.println("try block execution end");
          }
          catch(NullPointerException e)
          {
                  System.out.println("NullPointer exception is occurred");
          }
          catch(ArithmeticException e)
          {
                  System.out.println("Arithmetic exception is occurred");
          }
          catch(ArrayIndexOutOfBoundsException e)
          {
                  System.out.println("ArrayIndexOutOfBoundsException is occurred");
          }

          y=a/(b+c);
          System.out.println("Y="+y);
  }
}
```

Output

```
try block execution begin
Arithmetic exception is occurred
Y=1
```

❖ throw keyword in Java:

- System generated exceptions are automatically thrown by the Java runtime system. To manually throw an exception 'throw' keyword is used.
- The throw keyword in Java is used to explicitly throw an exception from a method or any block of code.
- **Syntax** of throw keyword is given below

```
throw ExceptionObject;
```

- **Example:**

  In this example, we have created the validate method that takes integer value as a parameter. If the age is less than 18, we are throwing the ArithmeticException by using throw keyword otherwise print a message 'You are eligible for voting'.

```
class ThrowDemo
{

  void validate(int age)
  {
          if(age<18)
          {
                  throw new ArithmeticException("Not valid Age for voting");
          }
          else
          {
                  System.out.println("You are eligible for voting");
          }
  }
  public static void main(String args[])
  {
          ThrowDemo d1=new ThrowDemo();
          d1.validate(16);
  }
}
```

Output

```
Exception in thread "main" java.lang.ArithmeticException: Not valid Age for voting
        at ThrowDemo.validate(Testtrycatch1.java:8)
        at ThrowDemo.main(Testtrycatch1.java:18)
```

Note: - No need to write the output in exam

❖ throws keyword in Java:

- throws is a keyword in Java which is used to indicate that this method might throw one of the listed type exceptions.
- Below is the general form of a method which includes a throws clause:

```
returntype method_name(parameters) throws exception_list
{
  //method body
}
```

- The throws do the same thing that try-catch does but there are some cases where you would prefer throws instead of try-catch. For example: Let's say we have a method myMethod() that has statements that can throw either ArithmeticException or NullPointerException, in this case you can use try-catch as shown below:

```
void myMethod()
{
  try
  {
            // Statements that might throw an exception
  }
  catch (ArithmeticException e)
  {
            // Exception handling statements
  }
  catch (NullPointerException e)
  {
            // Exception handling statements
  }
}
```

- But suppose you have several such methods that can cause exceptions, in that case it would be difficult to write these try-catch for each method. The code will become unnecessary long and will be less-readable.

- One way to overcome this problem is by using throws keyword: declare the exceptions in the method using throws and handle the exceptions where you are calling this method by using try-catch.

```
void myMethod()throws ArithmeticException, NullPointerException
{
            // Statements that might throw an exception
}
```

- **Example**

```
class ThrowsDemo
{
  void myMethod(int num)throws ArithmeticException, NullPointerException
  {
     if(num==1)
        throw new ArithmeticException("ArithmeticException Occurred");
     else
        throw new NullPointerException("NullPointerException Occurred");
  }
  public static void main(String args[])
  {
           try
           {
                ThrowsDemo obj=new ThrowsDemo();
                obj.myMethod(1);
           }
           catch(Exception e)
           {
                System.out.println(e);
            }
  }
}
```

Output

```
java.lang.ArithmeticException: ArithmeticException Occurred
```

❖ **Difference between throw and throws in java:**

1. Throws keyword is used to declare an exception which works similar to the try-catch block. throw keyword is used to throw an exception explicitly.

2. throw keyword is followed by an instance of Exception class and throws is followed by exception class names.

3. throw keyword is used in the method body to throw an exception and throws keyword is used with the method to declare the exceptions that can occur in statement present in the method.

4. By using throw keyword we can throw one exception at a time but you can handle multiple exceptions by declaring them using throws keyword.

❖ User defined exception in Java:

- We know the different exception classes such as ArithmeticException, NullPointerException etc. but all these exception classes are predefined which will be executed when particular condition is occurred.
- For example, when you divide a number by zero it executes the ArithmeticException.
- In Java we can create our own exception class and throw that exception using throw keyword.
- These exceptions are known as user-defined exceptions.
- User-defined exception must extend Exception class.
- The exception is thrown by using throw keyword.

Example:

```
class MyException extends Exception
{
  Myexception(String msg)
  {
           super(msg);
  }
}

class ExceptionDemo
{

  public static void main(String args[])
  {
           try
           {
                //MyException m1=new MyException("Used defined Exception");
                //throw m1;
                  throw new MyException("Used defined Exception");

           }
           catch(MyException e)
           {
                System.out.println(e);
            }
  }
}
```

Output:

```
MyException: Used defined Exception
```

❖ Applet:

- Applet is a small Java program which runs on internet.
- An applet is a Java program that runs in a Web browser.
- Applet is a like application program which can perform Arithmetic operations, display graphics, play sounds, accept user inputs, create animation and play interactive games.
- An Applet class does not have any main() method.
- We can create the applet by extending the java.applet.Applet class.
- To run the applet on internet, we need to add it within HTML page.
- There are two types of Applet:
  1. Local Applet
  2. Remote Applet

1. Local Applet:

- Applet which is developed locally and stored in a local machine is known as Local Applet.
- Execution of local applets doesn't require any internet connection.
- The web page will search the local system directories, find the local applet and execute it.
- Specifying a Local Applet:

```
<applet codebase="path" code="NewApplet.class" width=120 height=120 >
</applet>
```

- In above example the codebase attribute specifies a path name on your system for the local applet where that applet's code is present.

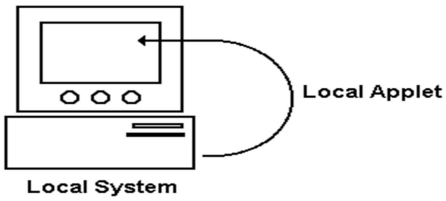


Fig. Loading local Applet from Local computer

2. Remote Applet:

- An applet which is developed by someone else and stored on a remote computer is called as Remote Applet.
- Execution of Remote applets internet connection is required.
- For locating and loading the remote applet, the URL must be specified in the codebase attributes of the HTML code.
- Specifying a Remote Applet:

```
<applet codebase="URL path" code="NewApplet.class" width=120 height=120 >
</applet>
```

- Below diagram shows, how the remote applet is loading and executing from remote system.

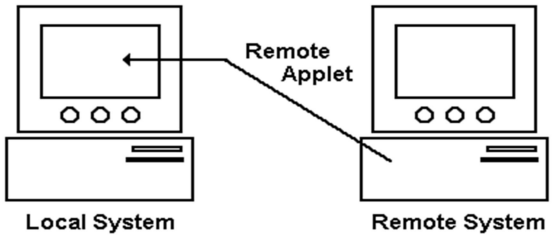


Fig. Loading Remote Applet from Remote computer

Advantages of Applet:

1. Applet works at client side so less response time.
2. Applets are platform independent.
3. Applets increase interactivity for users.
4. Database integration is another important advantage of applets.
5. Quick Execution

Disadvantages of Applet:

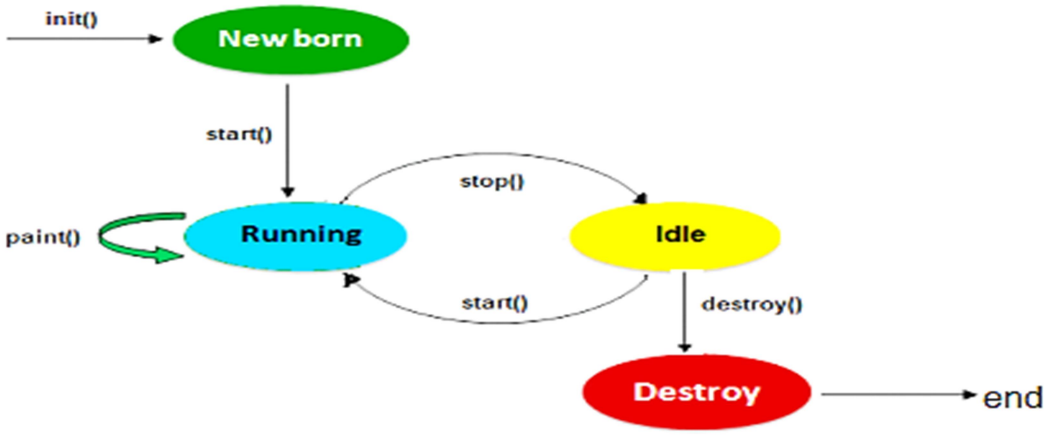
1. Applet cannot run independently.
2. Browser is required to run an applet

❖ Difference between Applet and Application

Applet	Application
1) Applet is a Small Java Program.	Application is a Large Java Program.
2) Applet is not a fully featured application programs.	Application is a fully featured program.
3) Applet program runs on client Browser	Application program can be executed on standalone computer system.
4) Applet programs are portable and It can be executed by any JAVA supported browser.	Need JDK, JRE, JVM installed on client machine.
5) Applet program do not contain main() method	Every application program contain main() method
6) Applet program cannot run independently	Application program can run independently
7) Applet Requires highest security.	Does not require any security.
8) Applet cannot access anything on the system except browser's services.	Can access any data or software available on the system

❖ Applet Life Cycle:

- Following diagram shows the life cycle of Applet.



Applet life cycle consists of 4 states which are given below:

1. New Born/Initialization State
2. Running State
3. Idle/stopped State
4. Dead/Destroy state

New Born State:

- When applet is first loaded then it enters into New Born /Initialization state by calling init() method.
- When Applet is born then it can do creation of objects, setting up initial values, Loading images or fonts, set up colors, etc.
- The Initialization of an applet is occurs only once in the life cycle of an Applet.
- Syntax of init() method:

```
public void init()
{
        //body of init method
}
```

Running State:

- When system calls the start() method of an applet class then it enters into **Running state.**
- This occurs automatically after initialization of an applet.
- Syntax of start() method:

```
public void start()
{
        //body of start method
}
```

- In this state, Applet actually is in running mode. Applet can calls to paint() method to draw any output on the screen. paint() method can takes Graphics class object as arguments.
- Syntax of paint() method:

```
public void paint(Graphics g)
{
        //body of paint method
}
```

Idle/Stopped State:

- When applet is stopped from running then it becomes Idle.
- Stopping of the applet is done automatically when we leaving the web page or we can done explicitly by calling stop() method.
- Idle state applet again come to the running state by calling start() method.
- Syntax of stop() method:

```
public void stop()
{
        //body of destroy method
}
```

Dead/Destroyed State:

- When applet is removed from the memory then it becomes dead.
- This will happen automatically when we exit from the web browser or we can done explicitly by calling destroy() method.
- Destroying of the applet is happen only once in the lifetime of the Applet.
- Syntax of destroy() method:

```
public void destroy()
{
        //body of destroy method
}
```

❖ How to run an Applet:

There are two different ways to run an applet code in Java Programming language.

1. Executing the Applet by using Java-compatible web browser.
2. Using an Applet viewer

1. Executing the Applet by using Java-compatible web browser:

Following steps are required for executing the Applet by using Java-compatible web browser:

Step 1:

- First, we need to create the applet Java code and compile it by using normal **javac** command.
- Example of Applet code:

```
//AppletDemo.java file

import java.applet.*;
import java.awt.*;

public class AppletDemo extends Applet
{
          public void paint(Graphics g)
          {
                    g.drawString("Welcome to the world of Applet",150,150);
          }

}
```

Step 2:

- After that we need to create HTML file in the same directory where the applet Java code is present. Inside the body tag of the HTML file, we need to include the applet tag for loading the applet class file.
- Example of HTML file and saved as MyApplet.html:

```
<html>
<body>
<applet code="AppletDemo.class" width="300" height="300">
</applet>
</body>
</html>
```

Step 3: Open the MyApplet.html file by double clicking on it and then you will able to see the output of the applet code.

2. Executing the Applet by using an Applet viewer:

Following steps are required for executing the Applet by using an Applet viewer

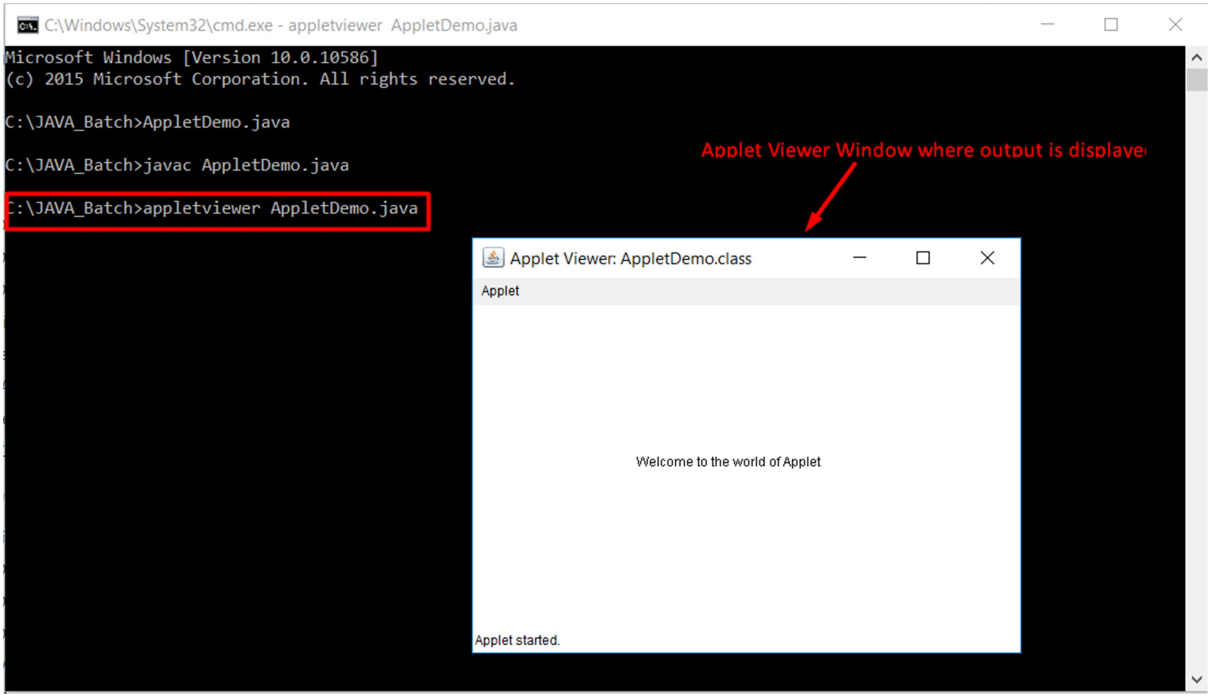
Step 1: First, we need to create an applet that contains applet tag in comment and compile it

Example of Applet code:

```
//AppletDemo.java file

import java.applet.*;
import java.awt.*;

public class AppletDemo extends Applet
{
            public void paint(Graphics g)
            {
                        g.drawString("Welcome to the world of Applet",150,150);
            }

}
/*
<applet code="AppletDemo.class" width="300" height="300">
</applet>
*/
```

Step 2: After that run it by using appletviewer command: **appletviewer First.java**

```
c:\>javac AppletDemo.java
c:\>appletviewer First.java
```



❖ **Applet Tag:**

- The pair of <APPLET> and </APPLET> tag is included into the body section of the HTML code.
- The <APPLET> tag is used to mention the name of the applet to be loaded and it tells the browser how much space is required to applet.
- **Syntax:**

```
< APPLET
    [CODEBASE = codebaseURL]
    CODE = Applet_File_Name
    [ALT = alternate_Text]
    [NAME = appletInstance_Name]
    WIDTH = pixels
    HEIGHT = pixels
    [ALIGN = alignment]
    [VSPACE = pixels]
    [HSPACE = pixels]
>
. . .

[< PARAM NAME = appletParameter1 VALUE = value1 >]
[< PARAM NAME = appletParameter2 VALUE = value2 >]
. . .

</APPLET>
```

Where:

CODEBASE = codebaseURL

This optional attribute specifies the base URL of the applet -- the directory or folder that contains the applet's code. If this attribute is not specified, then the document's URL is used.

CODE = Applet_File_Name

This required attribute gives the name of the file that contains the applet's compiled Applet code.

ALT = alternate_Text

This optional attribute specifies any text that should be displayed if the browser understands the APPLET tag but can't run Java applets.

NAME = appletInstance_Name

This optional attribute specifies a name for the applet instance, which makes it possible for applets on the same page to find (and communicate with) each other.

WIDTH = pixels
HEIGHT = pixels

These required attributes give the initial width and height (in pixels) of the applet display area.

ALIGN = alignment

This required attribute specifies the alignment of the applet. The possible values of this attribute are the same (and have the same effects) as those for the IMG tag: left, right, top, middle, bottom.

VSPACE = pixels
HSPACE = pixels

These optional attributes specify the number of pixels above and below the applet (VSPACE) and on each side of the applet (HSPACE).

< PARAM NAME = appletParameter1 VALUE = value >

PARAM tag is used to retrieve the user inputs to the applet code by using getParameter() method.

Example:

Following applet tag specifies the minimum requirements to put applet code on a web page.

```
< APPLET
    CODE = HelloJava.class
    WIDTH = 300
    HEIGHT = 300 >

</APPLET>
```

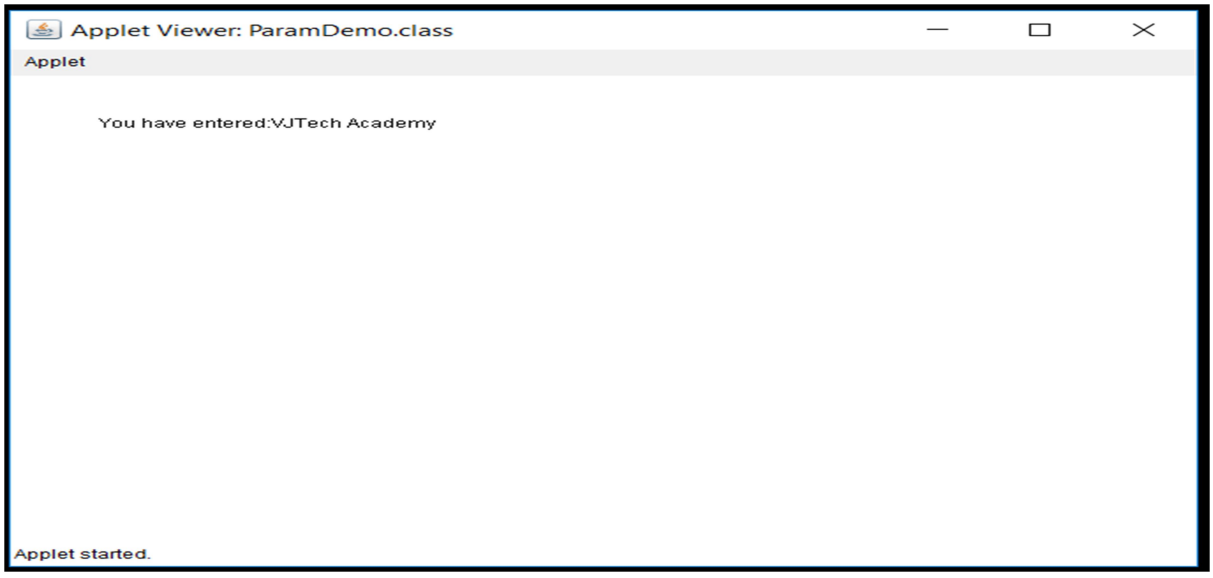

❖ **Parameter Passing to Applet or PARAM Tag:**

- The HTML <param> tag is used for passing parameters to an applet code.
- By using <param> tag we can pass the input values to the Java applet code.
- <param> tag contains two main attributes name and value.
- Applet code can retrieve the value which is associated with the name by using getParameter() method.
- Parameters are passed to the applet code when its loaded and inside the init() method we can retrieve the values of param tag.
- <param> tag needs to be included inside the html code with the <applet> tag.

Example:

```
import java.applet.*;
import java.awt.*;
public class ParamDemo extends Applet
{
  String msg;
  public void init()
  {
          msg=getParameter("String");
  }
  public void paint(Graphics g)
  {
          g.drawString("You have entered:"+msg,50, 50);
  }
}
/*
<applet code="ParamDemo.class" width="300" height="300">
<param name="String" value="VJTech Academy">
</applet>
*/
```

Output:



❖ **drawString() method of Graphics Class in Java:**

- drawstring() method is one of the most useful method of Graphics class to display the text messages on the window.
- **Syntax:**

```
drawString(String msg, int X ,int Y);
```

Where:

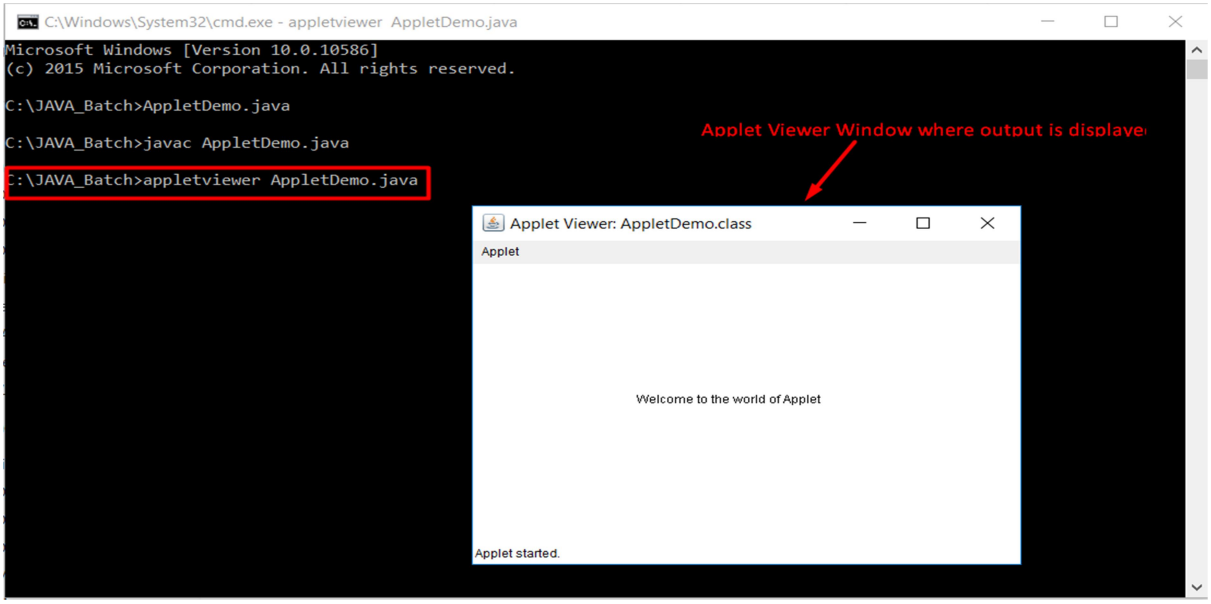
msg = msg is the string that can be displayed on the screen.

X, Y = X and y axis parameter value of the screen.

Example:

```
//AppletDemo.java file

import java.applet.*;
import java.awt.*;

public class AppletDemo extends Applet
{
                  public void paint(Graphics g)
                  {
                           g.drawString("Welcome to the world of Applet",150,150);
                  }

}
/*
<applet code="AppletDemo.class" width="300" height="300">
</applet>
*/
```

Output:



❖ drawLine() method of Graphics Class in Java:

- This method is used to draw the line on the output window.
- The drawLine() method takes four parameters of starting point x and y coordinates and ending point x and y coordinates.
- **Syntax:**

```
drawLine(int X1, int Y1, int X2, int Y2);
```

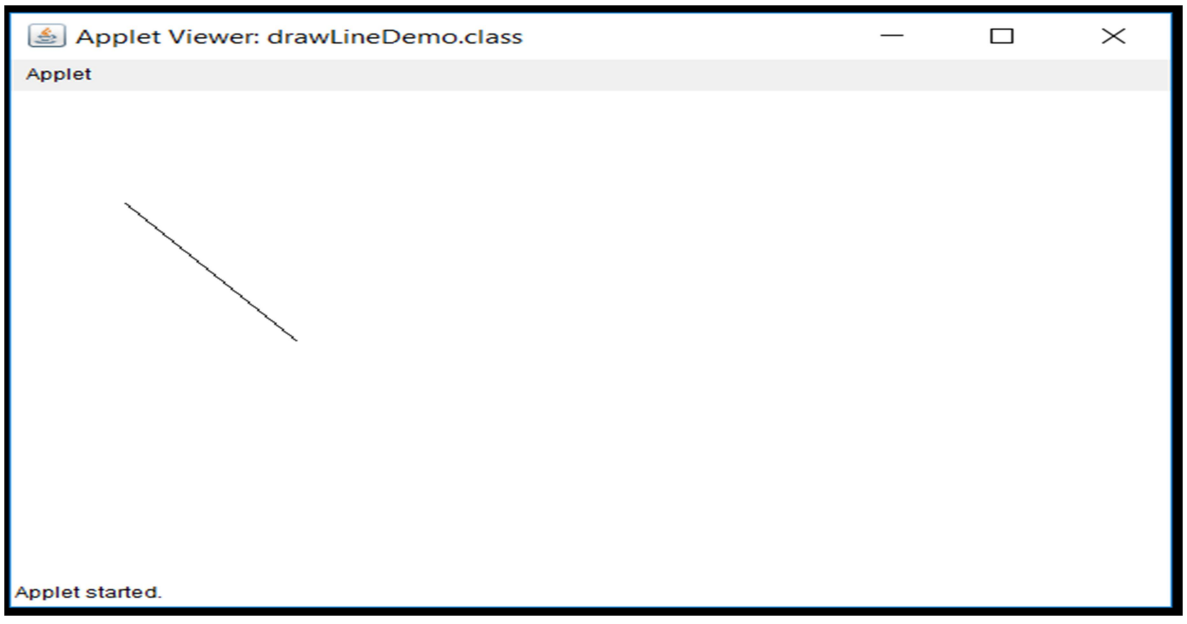
Where:

X1 and Y1 are starting point coordinates and X2 and Y2 are ending point coordinates of the line.

Example:

```
//drawLineDemo.java file

import java.applet.*;
import java.awt.*;

public class drawLineDemo extends Applet
{
            public void paint(Graphics g)
            {
                        g.drawLine(60,90,150,200);
            }

}
/*
<applet code="drawLineDemo.class" width="300" height="300">
</applet>
*/
```

Output:



This method takes four parameters, the starting coordinates (60, 90) and ending coordinates (150, 200) of the line to be drawn.

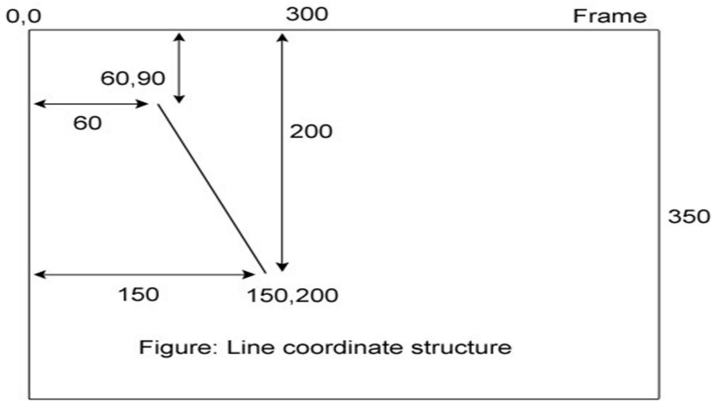


Figure: Line coordinate structure

❖ drawOval() method of Graphics Class in Java:

- This method is used to draw the circle and ellipse on the output window.
- The drawOval() method takes four parameters of starting point x and y coordinates and width and height of the circle or ellipse.
- If width and height parameters values are same then drawOval() method will draw the circle otherwise it will draw the ellipse.
- fillOval() method also draw the circle and ellipse but it's interior area is filled up.
- **Syntax:**

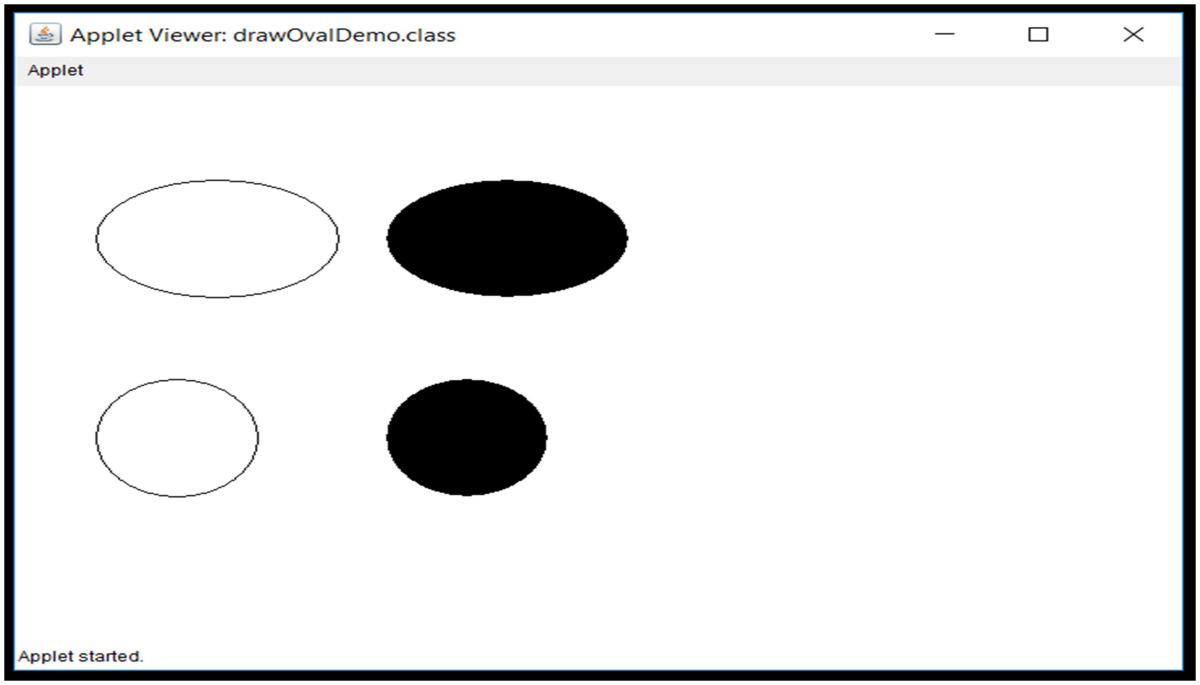
```
drawOval(int X, int Y, int Width, int Height);
fillOval(int X, int Y, int Width, int Height);=
```

Example:

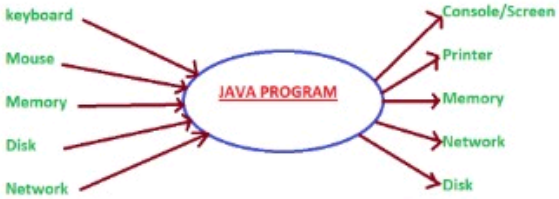
```
//drawOvalDemo.java file

import java.applet.*;
import java.awt.*;

public class drawOvalDemo extends Applet
{
                    public void paint(Graphics g)
                    {
                              g.drawOval(50,80,150,100);
                              g.fillOval(230,80,150,100);

                    }

}
```

```
/*
<applet code="drawOvalDemo.class" width="300" height="300">
</applet>
*/
```

Output:



Managing Input/Output Files in JAVA



1) System.in (Input Stream)
2) System.out (Output Stream)
3) System.err (Error Stream)

java.io package

Streams:

- Collection of continues group of data which will travel from one point to another point is known as Stream.

(a) Reading data into a program.

(a) Writing data to destination

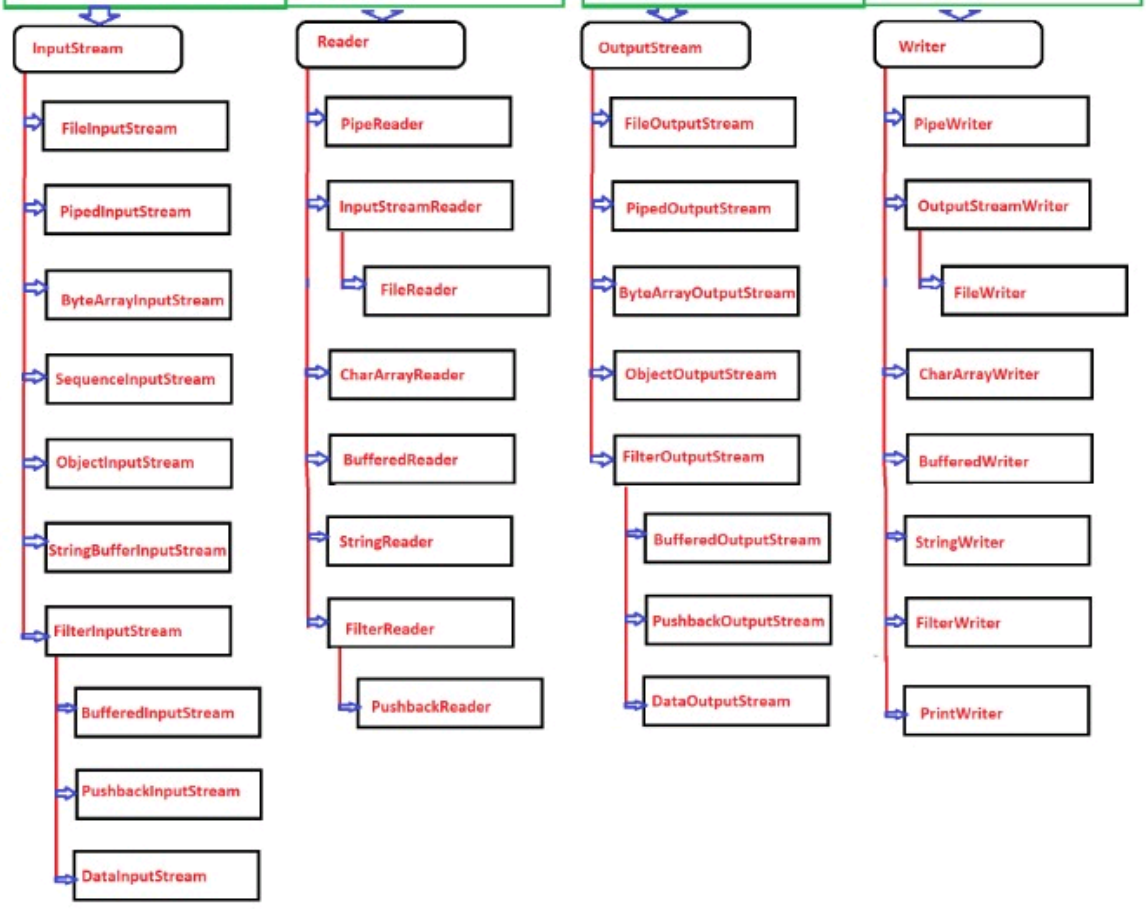
VJTech Academy

Byte Stream Class	Character Stream Class
1) Provides support for handling IO operations on byte.	1) Provides support for handling IO operations on characters.
2) Two types: i - Input Stream Classes ii - Output Stream Classes	2) Two types: i - Reader Classes ii - Writer Classes
3) Input stream classes are used to read 8 bit byte data.	3) Reader stream classes are used to read character from files.
4) Output stream classes are used to write 8 bit byte data.	4) Writer stream classes are used to write character into files.
5) We can translate character stream into byte stream with OutputStreamWriter.	5) We can translate byte stream into character stream with InputStreamReader.

VJTech Academy

InputStream Class	Reader Class
1) InputStream classes are used to read 8 bit bytes data.	1) Reader class is used to read 16 bit unicode character stream.
2) InputStream classes are byte-oriented.	2) Reader class is character oriented.
3) Constructor: InputStream()	3) Constructor: Reader() Reader(Object lock)
4) Methods: i) read() - Reads the next byte of data ii) read(byte b[]) - Read some bytes from input stream & stored them into byte array b. iii) read(byte b[],int m,int n) - Read n bytes data from input stream starting from mth byte & stored them into byte array b. iv) close() - Closes the input stream. v) skip(n) - skips over n bytes from input stream vi) available() - No of bytes available	4) Methods: i) read() - Read single character ii) read(char c[]) - Read characters into an array. iii) read(char c[],int m,int n) - Read n characters data from input stream starting from mth character & stored them into char array c. iv) close() - Closes the input stream. v) skip(n) - skips over n characters from stream vi) reset() - Goes back to the beginning of the stream.

OutputStream Class	Writer Class
1)OutputStream classes are used to write 8 bit bytes data.	1) Writer class is used to 16 bit unicode character stream.
2)OutputStream classes are byte-oriented.	2) Writer class is character oriented.
3) Constructor: OutputStream()	3) Constructor: Writer() Writer(Object lock)
4) Methods: i) write() - Writes byte of data ii) write(byte b[]) - Writes byte array b to the output stream. iii)write (byte b[],int m,int n) - Write n bytes data from array b to output stream which is starting from mth byte iv) close() - Closes the output stream v) skip(n) - skips over n bytes from output stream	4) Methods: i) write(char c) - write character c ii) write(char c[]) - writes character array. iii) write(char c[],int m,int n) - write n characters from array c starting from m character. iv) write(String str)- writes string. v) close() - closes the writer stream. vi) flush() - flushes the output stream.



File Class:

- File is a predefined class which is present under java.io package.
- Constructor 1:
  File(String dirPath)
- Example:
  File f1=new File("C:\\VJTech");
- Constructor 2:
  File(String dirPath,String fileName)
- Example:
  File f2=new File("C:\\VJTech","abc.txt");

IO Exception Classes

- CharConversionException
- EOFException
- FileNotFoundException
- InterruptedIOException
- InvalidClassException
- InvalidObjectException
- UnsupportedEncodingException
- WriteAbortedException

Methods:

1) String getName() - return the name of the file.
2) String getParent() - return name of the parent directory.
3) long length() - return length of file
4) boolean exists() - return TRUE if file is exists otherwise FALSE
5) boolean isFile() - return true if invoking object is file.
6) boolean isDirectory() - return true if invoking object is directory.
7) boolean canRead() - return true if file can be read.
8) boolean canWrite() - return true if file can be written.
9) String getPath() - return path of the file.
10) boolean rename(File newName) - rename the file.
11) boolean delete() - return true if file is deleted successfully.
12) boolean setLastModified(long date) - Set the time of last modification of the file.
13) boolean setReadOnly() - Makes the file read only.
14) boolean createNewFile() - Creates new File.
15) boolean mkdir() - create directory.

Our coaching class mobile app link:

https://play.google.com/store/apps/details?id=in.vjtechacademy.android

Our coaching class website link:

https://www.vjtechacademy.in/

See our google reviews